Digital Control Architecture with Tight Voltage Regulation-VHDL implementation

Final Project for EE533
Professor : Dr. Rachid Beguenane

Xiang Zhang

Content

Application and Background

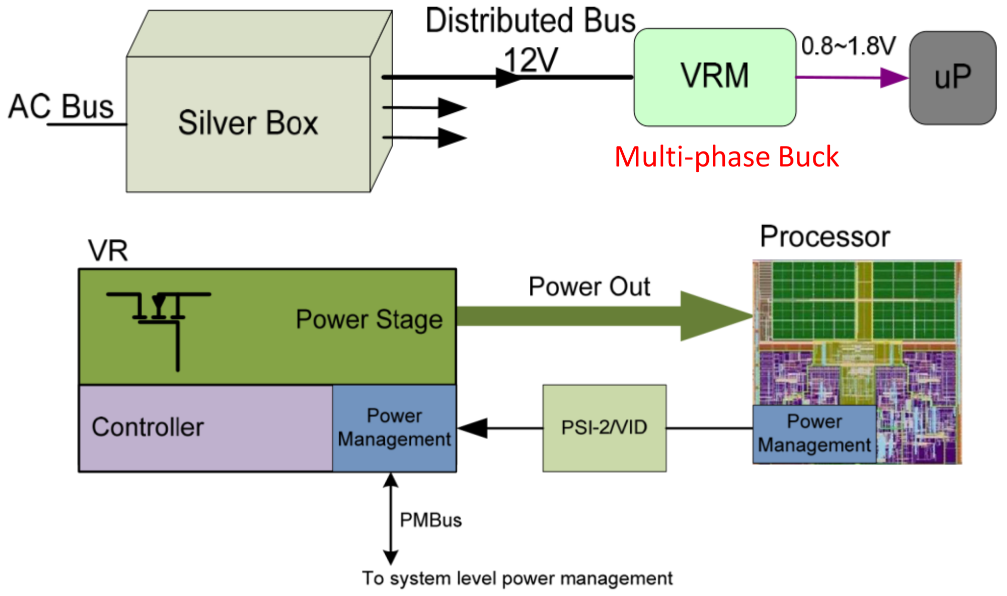
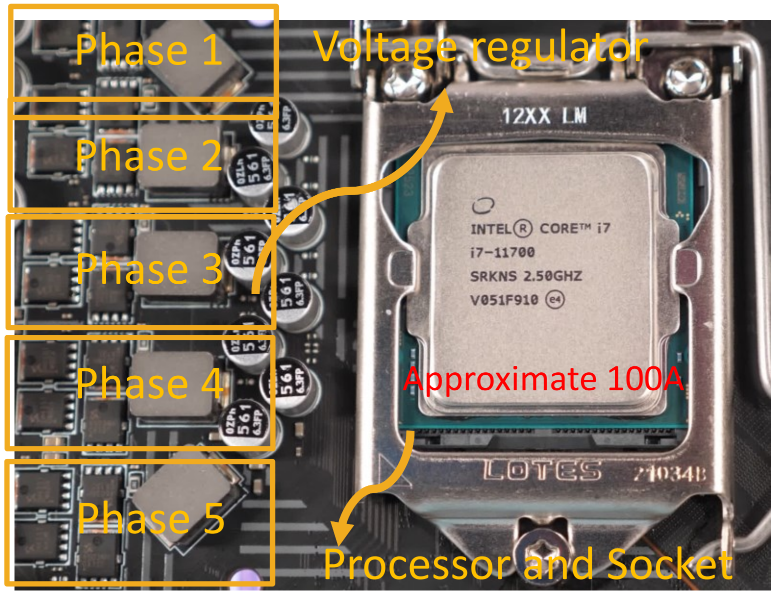


Illustration of Power management for modern processors

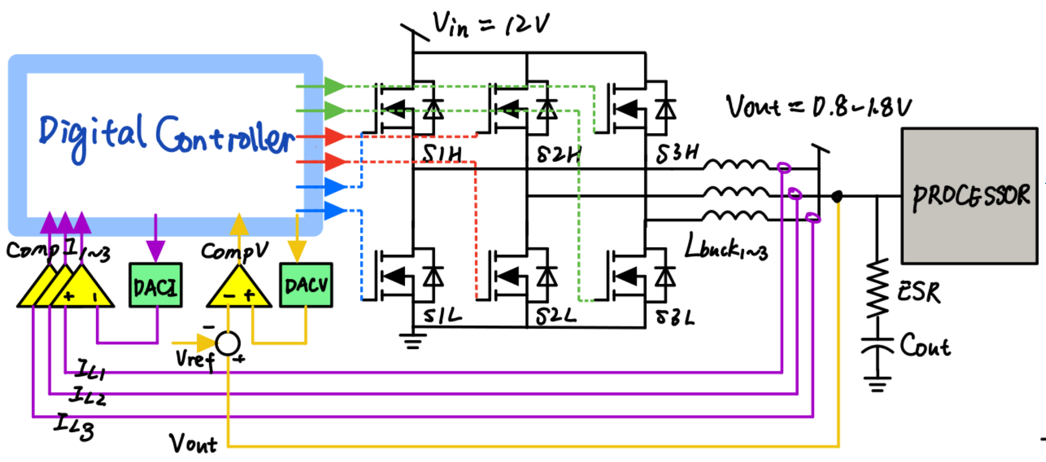
$$Power = Constant * Frequency * Voltage^2$$



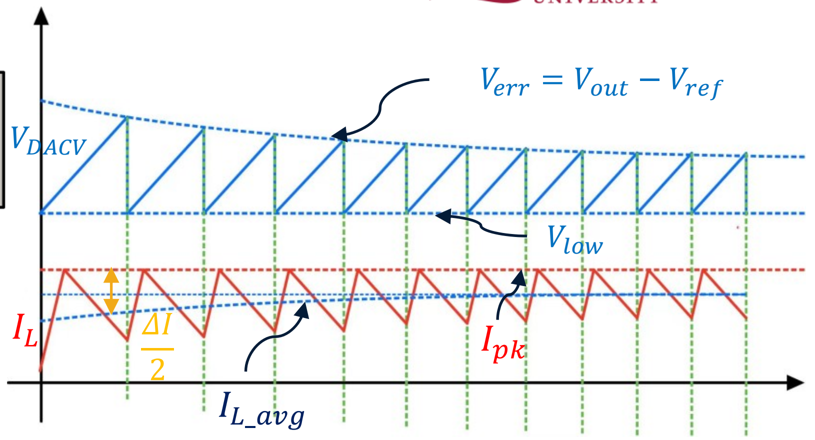
Voltage regulator of modern processor

- In some area, voltage tracking is very strict, requiring high dynamic responds, e.g. modern computer processors' voltage regulator;
- Dealing with different working-load, voltage is reconfigurable , e.g. tile up/down the voltage of the processor for overclocking to obtain better performance, intel turbo-boost;
- Multi-phase and interleaving techniques is required for all conditions

Digital Control System Overview



Simplified block diagram of the control architecture



Current and voltage waveform during small load transient

- Most digital control techniques use high resolution internal ADC (Analog-to-digital converter) and comparator , which results in a large core size and a high cost;
- Proposed Technique use external DAC and analog comparator, cost-friendly;
- Digital controller combines **current programmed control** and **variable frequency operation**

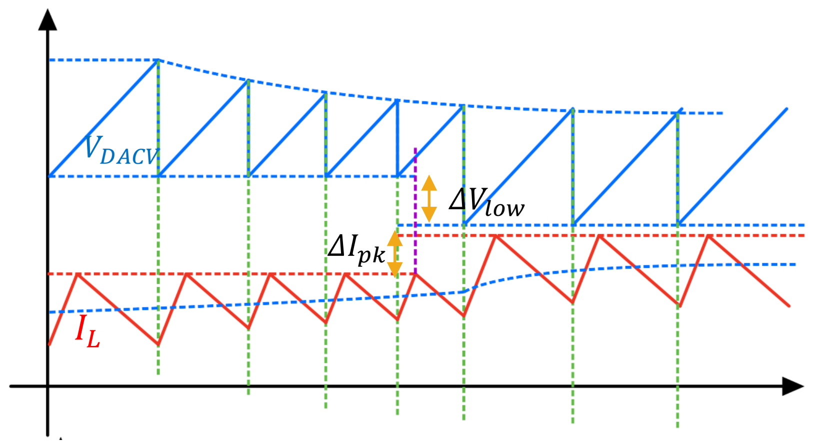
In small load transient, converter works in variable frequency operation

$$I_{L\_avg} = I_{out} = I_{pk} - \frac{\Delta I}{2} + \tau \frac{V_{out}(1-D)}{LD} = I_{pk} - \frac{V_{out}(1-D)}{2L\, f_s} + \tau \frac{V_{out}(1-D)}{LD}$$

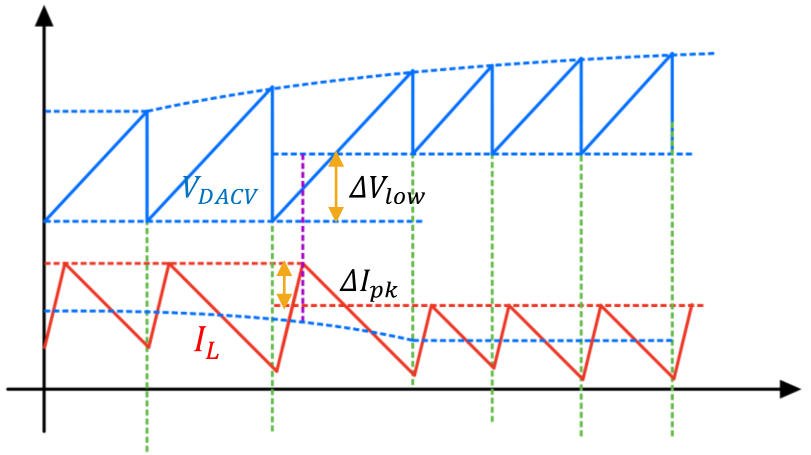
[1] S. Saggini, M. Ghioni, and A. Geraci, "An innovative digital control architecture for low-Voltage, high-current DC-DC converters with tight voltage regulation," *IEEE Transactions on Power Electronics*, vol. 19, no. 1, pp. 210–218, Jan. 2004, doi: 10.1109/TPEL.2003.820543.

Digital Control System Overview

In large load transient, converter works in current programmed control



Current and voltage waveform during large heavier load transient



Current and voltage waveform during large lighter load transient

$$V_{out} = V_{ref} - R_{droop} * I_{out}$$

- Under large load transient, controller introduce **adaptive voltage positioning techniques**: modifying the reference current and reference voltage simultaneously to improve dynamic voltage regulation performance, at this moment, the references is not fixed but load-dependent;
- The optimal value of R_{droop} is equal to the equivalent series resistance (ESR) of the output filter capacitor.

$$R_{droop} = \frac{\Delta V_{low}}{\Delta I_{pk}} = ESR$$

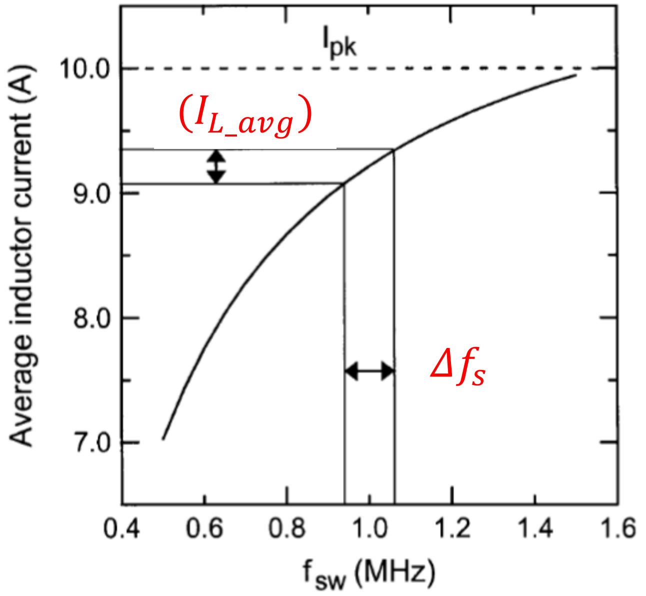
Determine the resolution of ADCs

Determine the current DAC (DACI)

According to the variable frequency operation principle, the average current varied with switching frequency (see page 4)

$$|\Delta I_{L\_avg}| = \frac{V_{out}(1-D)}{2L}\Delta T_s \geq LSB_{DACI}$$

- The main idea: DAC should be precise enough to make the least frequency variation detectable
- In other word: the least significant bit (LSB) of DAC should smaller enough to overlap the smallest current variation



Average inductor current as a function of switching frequency

$$LSB_{DACI} = \frac{I_{pk\_max}}{2^{n_{DACI}}}$$

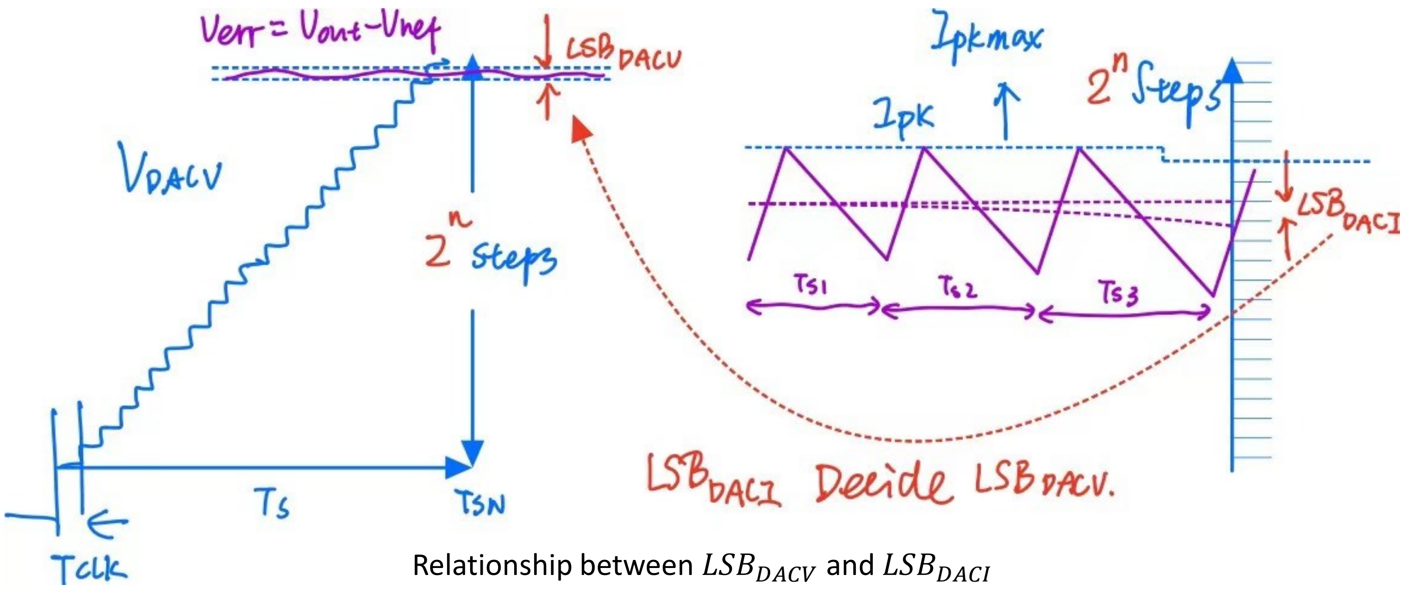
$$n_{DACI} \geq \log_2\left(\frac{15.4A * 1Mhz}{4A * (\pm 5\% * 1Mhz)}\right) + 1 = 7.266$$

Therefore, the resolution of current DAC can be calculated:

$$n_{DACI} \geq \log_2\left(\frac{I_{pk\_max} * T_{sN}}{\frac{V_{out}(1-D)}{2L}\Delta T_s}\right) = \log_2\left(\frac{I_{pk\_max} * f_{sN}}{\Delta I * \Delta f_s}\right) + 1$$

Determine the resolution of ADCs

Determine the volage DAC (DACV)



Relationship between LSB_{DACV} and LSB_{DACI}

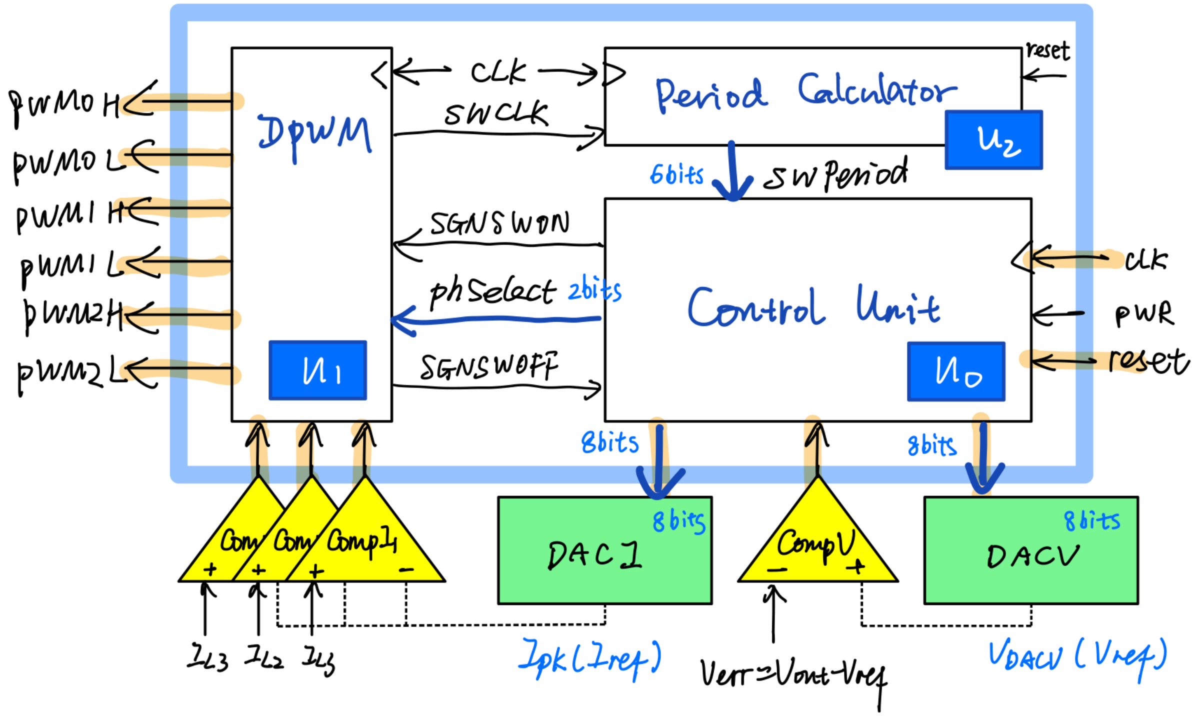
- The DACV is determined to satisfy the adaptive voltage positioning techniques, once the LSB_{DACI} is decided, the LSB_{DACV} is fixed accordingly.
- According to the adaptive volage positioning operation principle:

$$R_{droop} = ESR = \frac{\Delta V_{low}}{\Delta I_{pk}} = \frac{M_{update}}{N_{update}} * \frac{LSB_{DACV}}{LSB_{DACI}}$$

- When current variation is small (under ±5%), I_{pk_ref} keeps the same; while the variation is large (large load transient), update the refence current.

$$N_{update} = int[(T_{sN} - T_s) * f_{clk}]$$

Controller architecture Overview



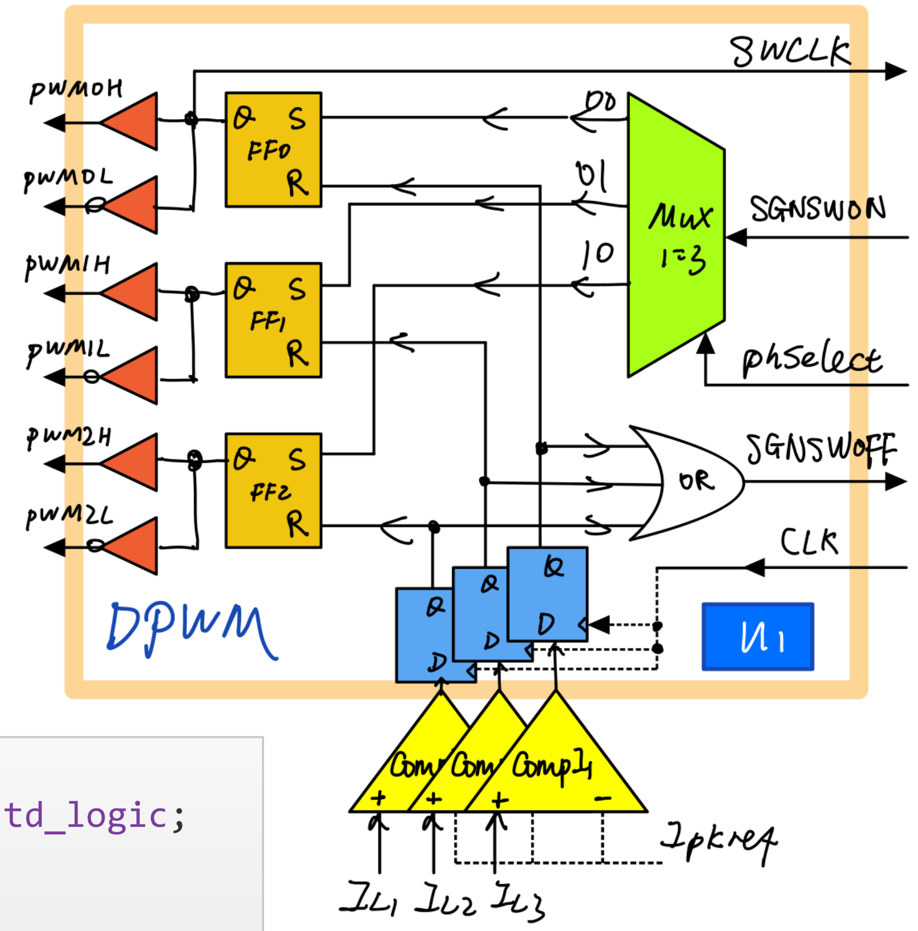
Digital Controller architecture proposed in the lecture

- The digital controller consist of three components, DPWM generator, Switching Period Calculator, and Control Unit;
- The DPWM is similar as the Datapath in traditional special purposed processor, and the Control Unit can be translated to a Finite State Machine.

[1] S. Pan and P. K. Jain, “A New Digital Adaptive Voltage Positioning Technique with Dynamically Varying Voltage and Current References,” *IEEE Transactions on Power Electronics*, vol. 24, no. 11, pp. 2612–2624, 2009, doi: 10.1109/TPEL.2009.2030807.

DPWM implementation

- D-Flip Flop transformed the unsynchronized COMPI signal to synchronized signal;
- SR latch controls the gate driver signals PWMnH/PWMnL;
- The Multiplexer changes switching phase in turns by phslect signal;
- The block returns SWCLK for switching period calculation and SGNSWOFF for state switch condition,

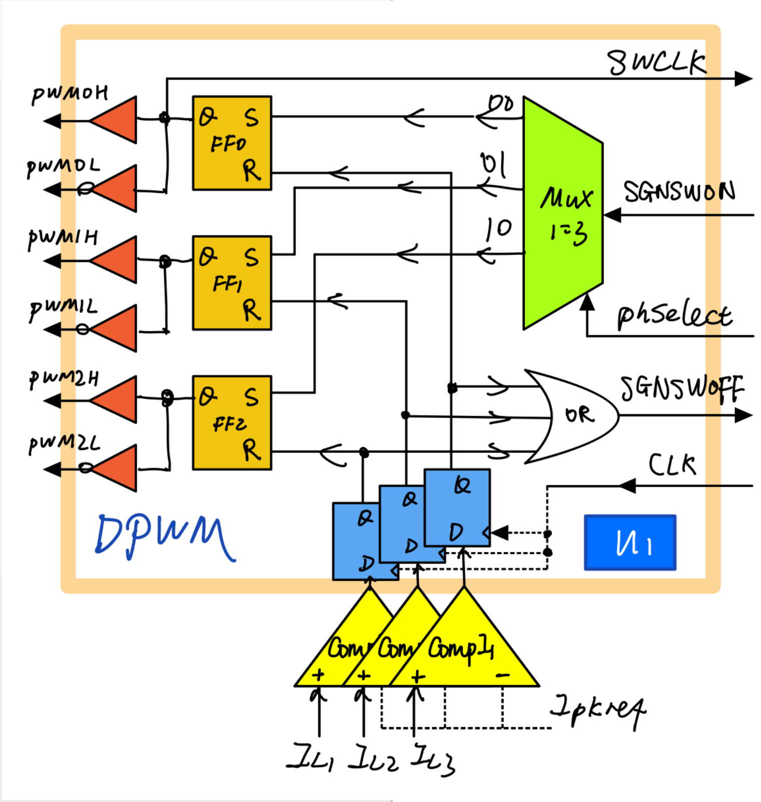


DPWM component structure

```
entity DPWM is
  port(SGNSWON, CompI1, CompI2, CompI3: in std_logic;
   PhSelect: in unsigned (1 downto 0);
   SGNSWOFF: out std_logic;
   SWCLK: out std_logic;
   PWM0H,PWM0L,PWM1H,PWM1L,PWM2H,PWM2L: out std_logic);
END DPWM;
```

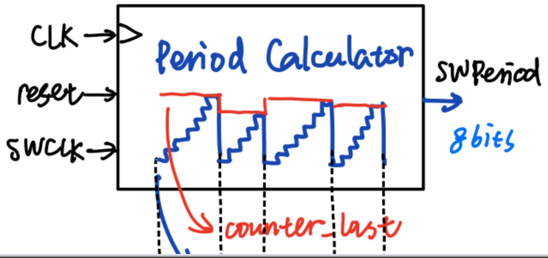
DPWM implementation

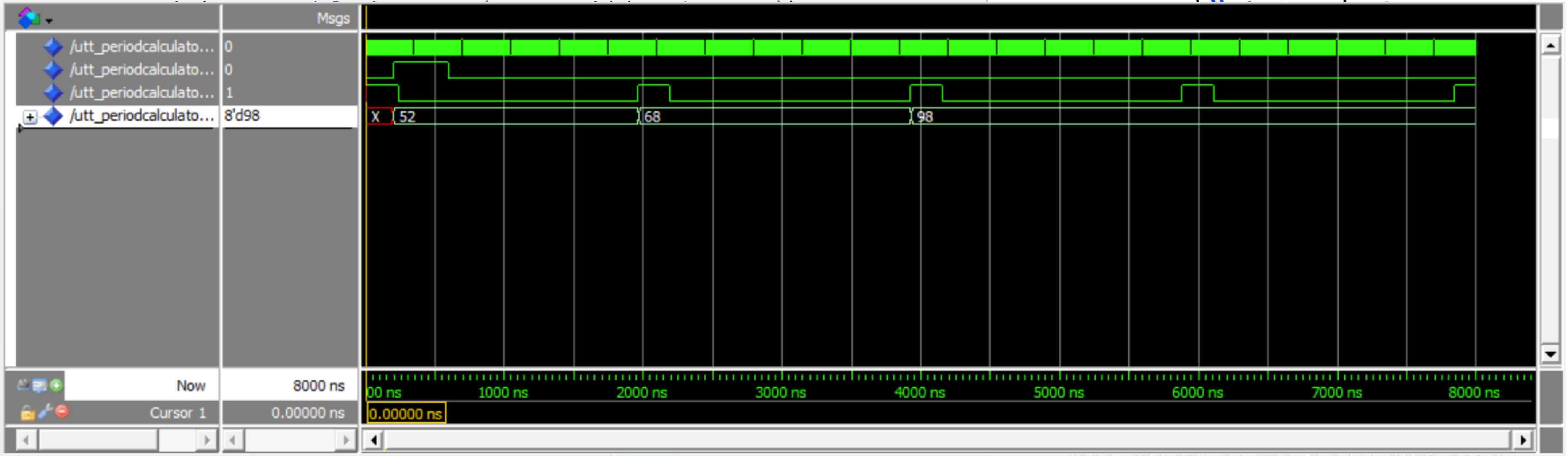
```vhdl
architecture structural OF DPWM IS
    signal Dset0,Dset1,Dset2,Drest0,Drest1,Drest2: std_logic;
    -- insert COMPONENT declarations
    begin
    SR0: entity SR_latch(sequential)
    port map(Dset0 => Dset, Drest0 => Drest,
    PWM0H => DQ, PWM0L => not DQ);

    SR1: entity SR_latch(sequential)
    port map(Dset1 => Dset, Drest1 => Drest,
    PWM1H => DQ, PWM1L => not DQ);

    SR2: entity SR_latch(sequential)
    port map(Dset2 => Dset, Drest2 => Drest,
    PWM2H => DQ, PWM2L => not DQ);

    DFF0: entity D_FlipFlop(sequential)
    port map(clk => clk, CompI1 => D, Drest0=> Q);

    DFF1: entity D_FlipFlop(sequential)
    port map(clk => clk, CompI2 => D, Drest1=> Q);

    DFF2: entity D_FlipFlop(sequential)
    port map(clk => clk, CompI3 => D, Drest2=> Q);

    MUX0: entity MUX_1to3(sequential)
    port map(PhSelect => MUXsel, Dset0 =>out0, Dset1 =>out1, Dset2 =>out1);

    SWCLK <= PWM0H;
    SGNSWOFF <= (Drest0 or Drest1 or Drest2);
end structural;
```



Period Calculator implementation

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity PeriodCalculator is
    generic(CounterWidth: integer:= 8;
            SWPeriod_norm: std_logic_vector:=B"00110100");

    port(reset:IN std_logic;
        CLK: IN std_logic;
        SWCLK:IN std_logic;
```





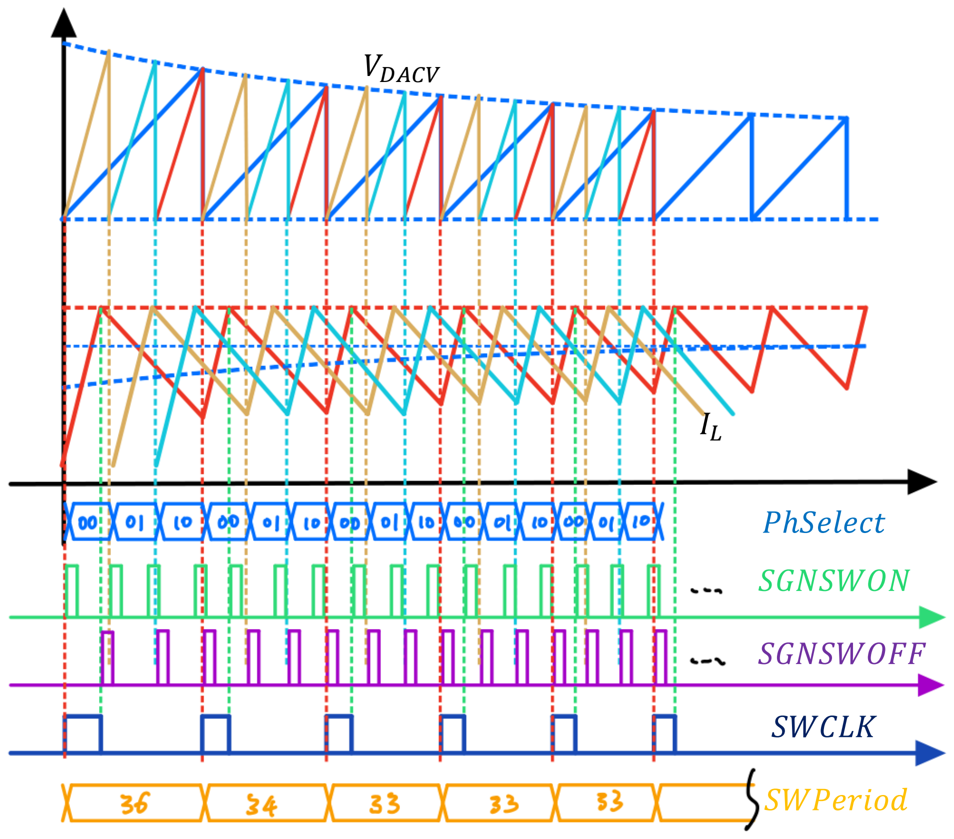
```
/utt_periodcalculato...   0
/utt_periodcalculato...   0
/utt_periodcalculato...   1
/utt_periodcalculato...   8'd98      X  52   68   98

Now       8000 ns     00 ns  1000 ns  2000 ns  3000 ns  4000 ns  5000 ns  6000 ns  7000 ns  8000 ns
Cursor 1  0.00000 ns  0.00000 ns
```

```
                        counter:=(others=>'0');
                else
                    if rising_edge(clk) then
                            counter:=counter+1;
                    end if;
                    SWPeriod <= counter_last;
                end if;
            end if;
    END PROCESS;
END behavioral;
```

turn-on signals (SWCLK)

- Main idea: use a counter_last to register the value just before the counter come back zero

Multi-phase control unit implementation



Key waveform of control architecture applied to a 3-phase buck converter

- In multi-phase operations, adding phase is equivalent to multiplying switching frequency, for m phase converter, the switching frequency f_s' is equivalent to $m * f_s$
- The difference is to add up a phSelect signal, for m phase, the phSelect signal should be at least $\log_2 m$ bits width

Minimum FPGA clock frequency

According to **Nyquist - Shannon Theorem**, the sampling should be at least two times the frequency bandlimited, that is, the least frequency variation we want to detect

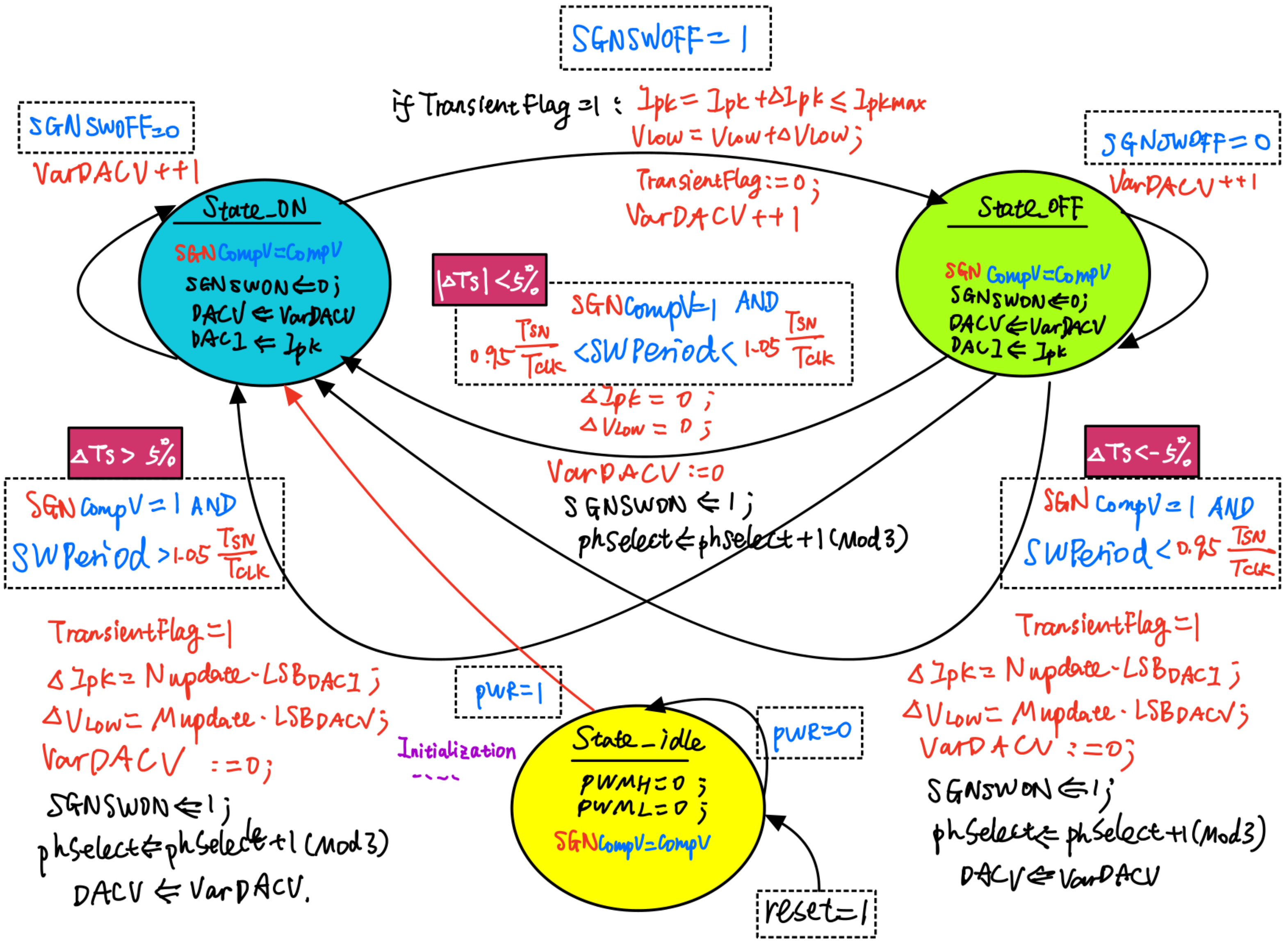
$$\frac{\Delta T_s}{T_{sN}} \geq m * \frac{2T_{clk}}{T_{sN}} \Rightarrow f_{clk} \geq \frac{m * 2f_{sN}}{\left(\frac{\Delta f_s}{f_{sN}}\right)}$$

The main defects of the control technique

Requires relatively high frequency FPGA, for a 3-phase 500kHz and $\pm 5\%$ frequency tolerance system, a FPGA speed of at least 30Mhz is required.

FPGA device (Xilinx, Virtex XCV 50) operated at 50MHz; four D/A converters 7bits

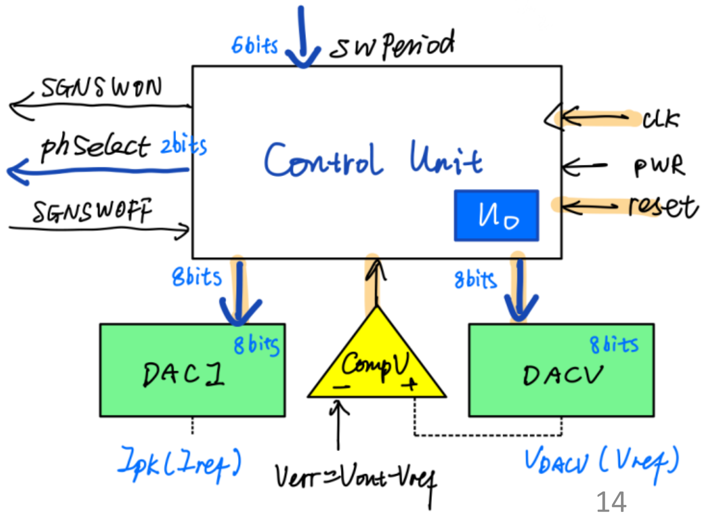
Control Unit with AVP technique FSM algorithm



Control Unit with AVP technique FSM algorithm

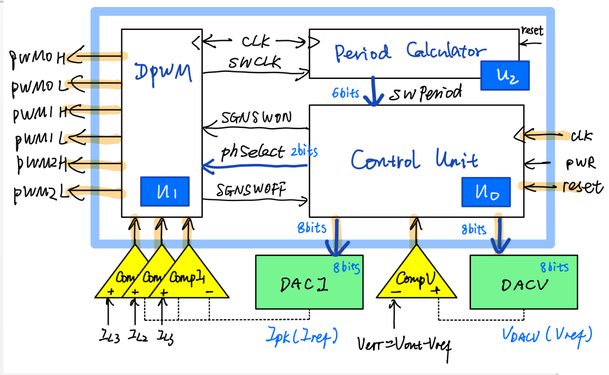
```vhdl
architecture algorithm of control_unit is
    type state_type is (state_on, state_off, state_idle);
    signal state: state_type;
    signal SGNcompV: std_logic:='0';

begin
    process(clk)
    begin
        if rising_edge(clk) then
            SGNCompV<=CompV;
        end if;
    end process;

    process(clk,reset)
        Variable VarSGNSWON :std_logic:='0';
        Variable VarDACV :std_logic_vector(DACWidth-1 downto 0):=B"00000000";
        Variable VarDACI :std_logic_vector(DACWidth-1 downto 0):=B"00000000";
        Variable TrasientFlag :std_logic:='0';
        Variable delt_Ipk,delt_Vlow :std_logic_vector(DACWidth-1 downto 0):=B"00000000";
        Variable VarPhSelect:std_logic_vector(1 downto 0):=B"00";
    begin
    if rising_edge(clk) then
        if reset = '1' then state <= state_idle;
        else
            case state is
                when state_idle =>
                    if PWR = '1' then
                        state <= state_on;
                    else state <= state_idle;
                    end if;
                when state_on =>
                    if rising_edge(SGNSWOFF)
                        then if TrasientFlag='1' then
                                VarDACV := VarDACV+delt_Vlow;
                                VarDACI := VarDACI+delt_Ipk;
                                TrasientFlag := '0';
                            end if;
                        state <= state_off;
                    end if;
                when state_off =>
                    if rising_edge(SGNCompV)AND(SWPeriod<B"01011111") then  --SWPeriod<95
                            TrasientFlag := '1';
                            TrasientFlag := '1';
                            state <= state_on;
                    else
                        if rising_edge(SGNCompV)AND(SWPeriod>B"01101001") then --SWPeriod>105
                            TrasientFlag := '1';
                            state <= state_on;
                        else
                          if rising_edge(SGNCompV)AND(SWPeriod>="01011111")AND(SWPeriod<=B"01101001") then
                                TrasientFlag := '0';
                                delt_Vlow:= B"00000000";
                                delt_Ipk:= B"00000000";
                                VarDACV:=B"00000000";
                                state <= state_on;
                            end if;
                        end if;
                    end if;
            end case;
          SGNSWON <= VarSGNSWON;
          PhSelect <= VarPhSelect;
          DACV <= VarDACV;
          DACI <= VarDACI;
        end if;
    end if;
    end process;
end algorithm;
```

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.ALL;


entity control_unit is
    generic(DACWidth: integer := 8;
            CounterWidth: integer:= 8);

    port(reset, clk, PWR: in std_logic;
        SWPeriod: in std_logic_vector(CounterWidth-1 downto 0);
        SGNSWON: out std_logic;
        SGNSWOFF: in std_logic;
        PhSelect: out std_logic_vector(1 downto 0);
        DACV,DACI: out std_logic_vector(DACWidth-1 downto 0);
        CompV: in std_logic);
end control_unit;
    end if;
    end process;
end algorithm;
```



Assembling the digital controller

```
architecture structural of processor is
---
-- COMPONENT declarations (DATAPATH & CONTOL UNIT)
signal  SWCLK, SGNSWON, SGNSWOFF: std_logic;
signal  PhSelect: std_logic_vector(2 downto 0);
signal  SWPeriod: std_logic_vector(CounterWidth-1 downto 0);

Begin
    U0: control_unit
        generic map (DACWidth => DACWidth, CounterWidth => CounterWidth)
        port map(reset=>reset,CLK=>CLK,PWR=>PWR,
        SWPeriod=>SWPeriod,
        SGNSWON=>SGNSWON,SGNSWOFF=>SGNSWOFF,PhSelect=>PhSelect,
        DACI=>DACI,DACV=>DACV,CompV=>CompV);

    U1: DPWM
        port map(SGNSWON=>SGNSWON,SGNSWOFF=>SGNSWOFF,PhSelect=>PhSelect,
        CLK=>CLK,SWCLK=>SWCLK,
        CompI1=>CompI1, CompI2=>CompI2, CompI3=>CompI3,
        PWM0H=>PWM0H,PWM0L=>PWM0L,PWM1H=>PWM1H,PWM1L=>PWM1L,PWM2H=>PWM2H,PWM2L=>PWM2L);

    U2: period_calculator
        generic map (CounterWidth => CounterWidth)
        port map(reset=>reset,CLK=>CLK,SWCLK=>SWCLK,SWPeriod=>SWPeriod);
end structural;
```



Conclusion

1. FPGA is ideal tools to achieve multi-model control techniques to improve the dynamic performance with very fast transient response

2. FPGA is flexible for fully reconfigurable digital architecture, as well as calculation sparse algorithm, but in calculation dense algorithm, large spaces and clock speed is required. one solution is to use FPGA to do signal process and PWM generation, and let the calculation process to general proposed processors

3. For multi-phase topology, FPGA is powerful for signal reproduction and synchronization because its concurrent feature