Design and Modeling of ADPLL with sliding-window for wide range frequency tracking

Chuan Shan, Dimitri Galayko, François Anceau

Design and Modeling of ADPLL with sliding-window for wide range frequency tracking

Chuan Shan, Dimitri Galayko, François Anceau
LIP6 - UPMC Sorbonne Universités
Paris, France
chuan.shan@lip6.fr, dimitri.galayko@lip6.fr, francois.anceau@lip6.fr

***Abstract*— An architecture of All-Digital Phase-Locked Loop (ADPLL) with sliding window for wide range frequency tracking is proposed to reduce energy consumption and to accelerate convergence. A synthesizable VHDL model is created for this circuit. Simulation and syntheses results demonstrate high performance of the new architecture.**

I. Introduction

PLL technique is widely applied in clock distribution network design. It can be used for global and local clock generation[1][2][3]. The structure of a typical PLL is presented in Fig. 1 [4]. A PFD detects the phase/frequency difference between the locally-generated clock and a reference clock, and generates a signed binary code. This code is then processed by a loop filter, so as to generate a control word for the Digitally Controlled Oscillator (DCO). The frequency divider is used to generate a clock with a frequency higher than the one at which the error phase information is processed.

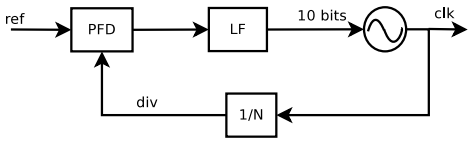

Fig. 1. Traditional PLL structure

A traditional ADPLL with PI filter has two disadvantages in the case where the DCO frequency range is wide, as required for the modern clocking and RF systems. One is long frequency acquisition time. The other is high power consumption of high-frequency multi-bit arithmetic operations related to the PI filter processing.

A common solution to this problem consists in separating the phase correction and frequency acquisition stages, devoting the LSBs of DCO control word to phase tracking (at high rate), and MSBs to frequency tracking (at low rate) [6]. Implicitly, this method divides DCO dynamic range into several segments. However, a problem of this approach is that boundaries of segments are predefined by the lengths of MSBs and LSBs. If the code corresponding to the reference frequency happens to be at the boundary between two segments, MSBs value will hop between two adjacent code, the MSBs must change with high frequency in order to achieve the phase tracking.

A new method with floating frequency segment is proposed in this article to solve this problem. As in some previous works [6], actual DCO control code is obtained as sum of a large coarse value and a small signed correction code. However, in the proposed architecture, the LSB of the coarse code and correction code have the same weight. Coarse frequency code is dynamically updated at a relatively slow frequency. A PFD and a filter calculate a signed correction code, which is added to coarse frequency code. It allows fast updating of DCO control word within a certain range around the coarse frequency. This structure implements a 4 bit range frequency window slowly sliding on 10 bits range. An update of the coarse frequency code provides an immunity to slow variations of the DCO initial frequency due to temperature, etc. This method provides a substantial economy of high frequency arithmetic operations, which results in a power saving.

This paper presents the proposed architecture in Section II. Simulation and synthesis results are shown in Section III.

II. New Architecture

The architecture of proposed PLL is shown in Fig. 2. A Reference Frequency Indicator (RFI) block, apart from the main PLL circuit, gives a code ($code_{ref}$) corresponding to the reference clock frequency. This code is then sent to PLLs in clock distribution network. Phase tracking is achieved by a 3-bit PFD and a 4-bit filter generating a 4-bit signed code based on the phase difference between two clocks. Frequency tracking is achieved by a mean filter, which receives a stream of 4-bit codes and calculates the average of eight most recently received ones. The sign of the average value ($+1$, -1, or 0) estimates the frequency relation between two clocks, and is used by the Coarse Frequncy Adjustment (CFA) block for coarse frequency code adjustment. The output code of CFA block plus the 4-bits filter output forms DCO control word.

The advantage of this structure is that CFA block can always update the coarse frequency (fc) of DCO according to the reference-local clock relation, and this adjustment is performed at a frequency lower than sampling frequency of system (div). PFD and filter, working at frequency div, tune DCO frequency around fc. This tuning process corrects phase error and at the same time works together with CFA to make sure that the reference frequency is always in the tuning interval. As the interval is relatively small, a big number of bits for PFD and filter is no longer necessary.

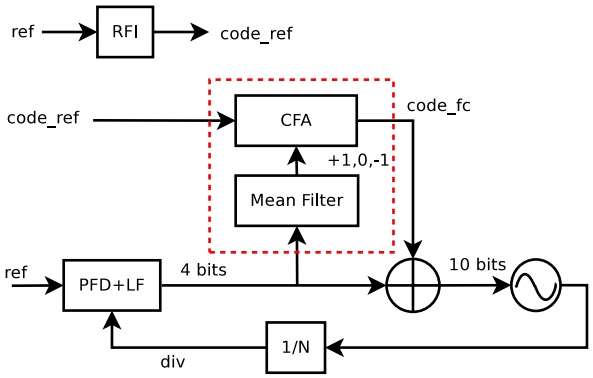

Fig. 2. Proposed architecture for PLL

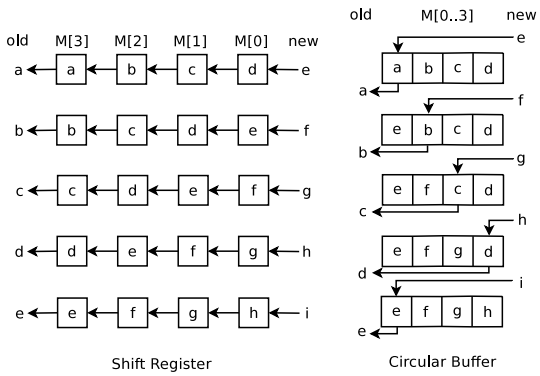

Fig. 3. Sliding window algorithm

A. Reference frequency indicator (RFI)

A RFI is an extra block for one PLL or a clock distribution network. It gives the system a 10-bits coarse estimation code of the reference frequency ($code_{ref}$) at starting stage. A RFI block can be implemented as a Look-Up Table (LUT), which takes reference frequency and temperature as inputs and gives corresponding code with certain precision very fast. The implementation of this block is not an emphasis of this paper.

B. Coarse frequency adjustment

The coarse frequency acquisition is achieved by the mean filter block and CFA block.

1) Mean filter: A mean filter uses a sliding window to calculate average value of last 8 inputs. The equation of a mean filter with input $x[i]$ at a given moment i is:

$$sum[i] = x[i-7] + x[i-6] + \cdots + x[i] \quad (1)$$

$$avg[i] = sum[i]/8 \quad (2)$$

where $sum[i]$ and $avg[i]$ are sum value and average value. If we define a signal new for the new input data and another signal old for the oldest data value at that moment, $sum[i]$ can also be represented by (3).

$$sum[i] = sum[i-1] - old[i] + new[i] \quad (3)$$

There are two ways to design a sliding window: shift register and circular buffer, whose principles are presented in Fig. 3.

A circular buffer [7] is a memory storing the last N output data of filter. An index points out which register stores the oldest data. The data stored in the register pointed by the current index is substituted with the new input. Instead of shifting the data, the index is shifted at each clock event. In this case, only one register is needed to be updated. A circular buffer requires less power than a traditional shift register.

For this reason, circular buffer is chosen to implement the sliding window of mean filter. To avoid conflicts, the mean filter works in two phases, one phase (S1) for reading the oldest data and subtracting it from sum, and the other phase (S0) for writing new data and adding it to last calculation result. Transitions of phases are sensitive to edges of clk signal, as shown in Fig. 5. A controller, implemented as a finite state machine (FSM), verifies transition conditions of working state, generates the write enable signal (WE) and new index for the circular buffer (idx), updates the input for ALU ($IN1$), and calculates the average value (avg) of sum and its sign avg_sign. The block diagram of the whole mean filter is presented in Fig. 4. TABLE. I details definition and output signal values of two different working states.

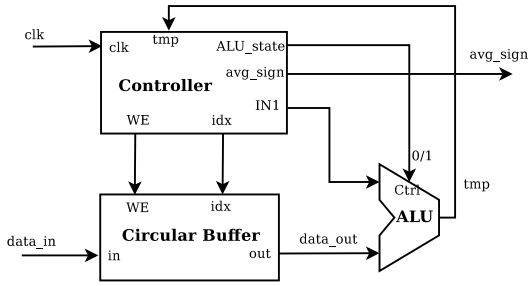

Fig. 4. Mean filter implementation

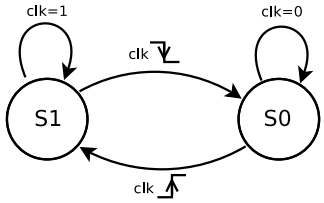

Fig. 5. State diagram of controller

TABLE I
CONTROL TABLE

clk	state	idx_n	WE	ALU_state
1	S1	$(idx_{n-1}+1) \bmod N$	0	1 (mode $\ominus$)
0	S0	idx_n	1	0 (mode $\oplus$)

The circular buffer storing last N output data of filter is implemented as an array of N registers (N=8 here) with a N-to-1 multiplexer and a 1-to-N demultiplexer. In writing mode, the value of idx signal defines which clock signal flips, and hence, which register stores the new $data\_in$ value. The data

update of each register is done only at the rising edge of its clock signal. In reading mode, the output always takes value of the register selected by idx. As shown in the chronograph (Fig. 7), $data\_in$ and idx are always prepared half clk period before the rising edge of WE.

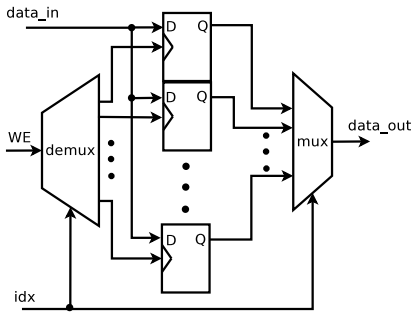

Fig. 6. Implementation of memory in mean filter

Block ALU is an adder/subtractor. Its working mode depends on $ctrl$ signal value. The output of ALU is either the intermediate calculation result or a new sum value:

$$tmp_{(s1)}[i] = sum[i-1] - old[i] \quad (4)$$

$$tmp_{(s0)}[i] = tmp_{(s1)}[i] + new[i] \quad (5)$$

Controller takes this temporary result (tmp), and prepares the input for new cycle calculation at the beginning of each state. The sign of average value avg_sign is obtained by controller based on ALU output.

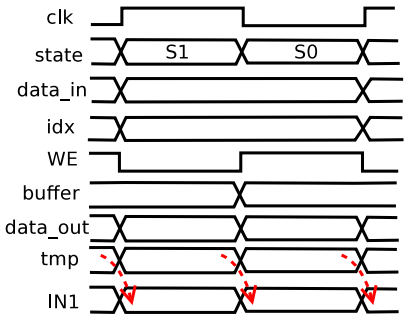

Fig. 7. Chronograph of mean filter

Some timing constraints have to be met for correct functionality of circuit. First, update of signal idx and subtraction operation should be carried out within the first state $S1$. Second, new income data is stored in proper register and an addition gives the new sum value, which should be finished before the end of the second state $S0$. Each state lasts half period of clk signal.

To verify the timing conditions, synthesis is done using Synopsys Design Compiler under a CMOS 65nm technology. Timing report shows that the work of $S1$ is finished in 2.38 ns and 2.87 ns for $S0$. There is a large positive slack in each state. Timing constraints are satisfied.

2) CFA: CFA updates the coarse frequency code by ± 1 or 0 at each cycle. To identify an eventual regular error on the coarse frequency, the mean filter needs to accumulate the output of the phase tracking blocks during several cycles. There is a trade-off between the coarse frequency tracking precision and the number of cycles taken into consideration: the latter defines the size of shift register and ALU in Fig. 4. A CFA structure with two integrators is implemented to relax the constraint in mean filter (Fig. 8). The first integrator accumulates mean filter output value. If there is overflow or underflow, this integrator is reset to 0. The second integrator, which is initialized to $code_{ref}$, takes the overflow/underflow value to adjust coarse frequency progressively.

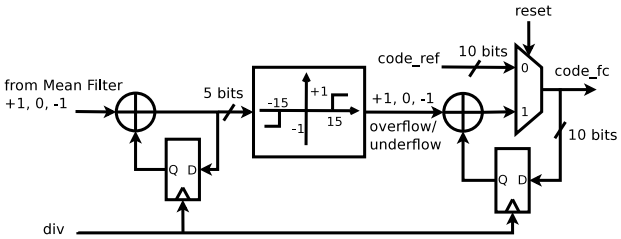

Fig. 8. CFA structure

C. Phase error correction

Phase error is corrected by a 3-bits traditional PFD and a 4-bits traditional PI filter [4]. The 4-bits output of PI filter is added to the output of CFA to form the DCO input word.

1) Drawbacks: The PLL transfer function in s-domain is:

$$H(s) = \frac{K_{PFD}K_{DCO}(K_p s + K_i)}{s^2 + sK_{PFD}K_{DCO}K_p + K_{PFD}K_{DCO}K_i} \quad (6)$$

where K_{PFD} and K_{DCO} are the gain of PFD and the gain of DCO respectively. K_p and K_i are the proportional and integral coefficients of the PI filter.

If we compare (6) with the common transfer function of 2nd order system, the damping factor ξ is obtained.

$$\xi = \frac{K_p}{2}\sqrt{\frac{K_{PFD}K_{DCO}}{K_i}} \quad (7)$$

As shown in Fig. 9, due to the dynamic range limit of a 3-bits PFD, there is saturation when phase error is larger than $3\Delta T_{TDC}$ or less than $-3\Delta T_{TDC}$. Hence the gain K_{PFD} is much smaller in the saturation region than the one in quasi-linear region. According to (7), this results in a relatively small damping factor ξ in the saturation region, which causes a relatively slow correction speed for large phase error.

2) Solution - Adaptive filter: According to (7), the damping factor ξ is function of K_{PFD}, K_{DCO}, K_p and K_i. Since K_{PFD} and K_{DCO} are defined by design specification, only the filter coefficients can be modified to compensate the effect of K_{PFD} diminution in saturation region. K_i is less effective than K_p, because K_i is under root and it has already a very small value in current design. If this value is reduced furthermore, the integral path performance is also reduced.

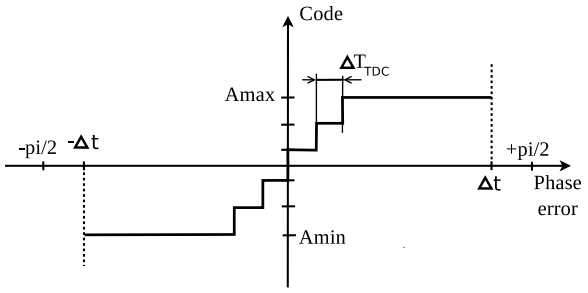

Fig. 9. Transfer function of 3-bits PFD

Hence, the solution is using an adaptive filter with a variable K_p instead of a regular PI filter in order to change damping factor of system on the fly when PFD works in different regions [8]. In this implementation, the value of K_p in saturation region is three times of that in quasi-linear region.

III. Comparison with traditional PLL

A. Functional Simulation results

Fig. 10 shows the phase error between the reference clock and clock generated by a regular ADPLL [5]. The reference frequency is 297.3 MHz at first, and it changes to 225 MHz since 10 us. It takes 13.5 us for PLL to be re-synchronize with the new frequency.

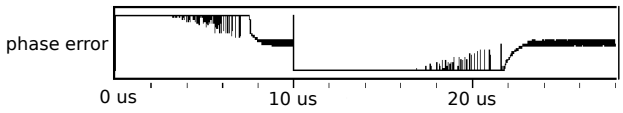

Fig. 10. Simulation of traditional architecture $Kp = 1, Ki = 15/2^{11}$

Fig. 11 presents result of a simulation of the new ADPLL structure presented in this paper but with a regular PI filter. Same initial condition and stimulus as last one are applied. The nominal code of PLL is reset with the new code after the change of frequency at 10 us. In this case, the re-convergence time is 2 us.

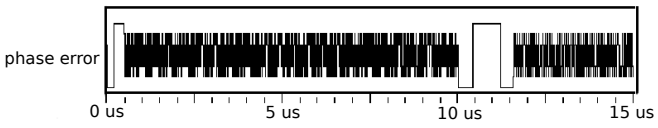

Fig. 11. Simulation of new architecture with regular PI filter $Kp = 1, Ki = 15/2^{11}$

If an adaptive filter is implemented in the model, the re-convergence time is shortened to 1 us (Fig. 12). As expected, the proposed PLL highlights a very high convergence speed.

B. Power consumption comparison

Syntheses using Synopsys Design Compiler are done for traditional architecture [5] and proposed architecture under ST Microelectronics CMOS 065 nm technology. The div signal is chosen to be at 125 MHz, which is 1/8 of PLL nominal frequency. TABLE II shows the power consumption of two architectures in μW. We can see that 37.4% of total power consumption is reduced by using the proposed architecture.

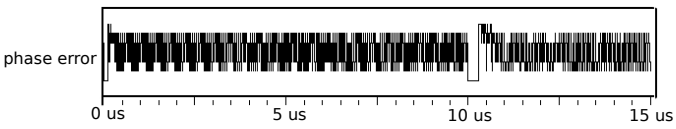

Fig. 12. Simulation of new architecture with adaptive filter $Kp = 1$ or 3, $Ki = 15/2^{11}$

TABLE II
POWER CONSUMPTION OF TWO ARCHITECTURES

cell	Traditional Architecture	Proposed Architecture
PFD	20.9	16.1
Loop Filter	51.0	12.4
Mean Filter	-	11.7
Incrementer	-	4.8
Total	71.9	45.0

IV. Conclusion

A new ADPLL architecture with sliding window for frequency tracking is proposed. A synthesizable VHDL model is developed for functionality validation and power consumption analysis. This architecture will be implemented in a clock distribution network in future work.

V. Acknowledgment

This work has been funded by the French National Agency of Research (ANR) under grant ANR-10-SEGI-014-01.

References

[1] Anderson, F.E. and Wells, J.S. and Berta, E.Z. *The core clock system on the next generation Itanium1 microprocessor*, IEEE International Solid-State Circuits Conference, 2002, pp. 146-453.

[2] Li, S. and Krishnakumar, A. and Helder, E. and Nicholson, R. and Jia, V. *Clock generation for a 32nm server processor with scalable cores*, IEEE International Solid-State Circuits Conference, 2011, pp. 82-83.

[3] Shan, C. and Zianbetov, E. and Javidan, M. and Anceau, F. and Terosiet, M. and Feruglio, S. and Galayko, D. and Romain, O. and Colinet, E. and Juillard, J. *FPGA implementation of reconfigurable ADPLL network for distributed clock generation*, IEEE International Conference on Field-Programmable Technology (FPT), 2011, pp. 1-4.

[4] E. Zianbetov et al., *Design and VHDL modeling of all-digital PLLs*, 8^{th} IEEE international NEWCAS conf., 2010, Montreal, QC, pp. 293-296

[5] Javidan, M. and Zianbetov, E. and Anceau, F. and Galayko, D. and Korniienko, A. and Colinet, E. and Scorletti, G. and Akre, JM and Juillard, J., *All-digital PLL array provides reliable distributed clock for SOCs*, IEEE ISCAS, 2011, pp. 2589-2592

[6] J. A. Thierno et al., *A Wide Power Supply Range, Wide Tuning Range, All Static CMOS All Digital PLL in 65 nm SOI*, IEEE JSSCC, vol. 43, no. 1, January 2008.

[7] Rose, G., *A stream cipher based on linear feedback over GF (2^8)*, Information Security and Privacy, Springer, 1998, pp. 135-146.

[8] Xiu, L. and Li, W. and Meiners, J. and Padakanti, R., *A novel all-digital PLL with software adaptive filter*, IEEE JSSCC, vol. 39, no. 3, 2004, pp. 476-483.