Greta Oto GNSS Receiver Design

Jun Mo

Globsky Technology Inc.

2021/5/20

Preface

As shown in the cover image, Greta Oto is a butterfly with transparent wings. The reason why I name this project is based on the following reason. About ten years ago, I started to design universal satellite navigation receiver architecture. At first, the purpose of this architecture was to use the simplest structure possible to realize a receiver suitable for multiple satellite signals. I named this architecture Butterfly to describe its lightness and flexibility. Based on the Butterfly architecture, various extensions were made based on new GNSS satellite signals. Now it is successfully adopted by several multi-system and multi-frequency GNSS receiver chips and go into production stage. The OPENGNSS project I released before is also based on such architecture. After releasing all the C models and part of the RTL of the baseband hardware of the OPENGNSS project, during the subsequent maintenance process, I found that due to the lack of specific product requirements, the OPENGNSS project is only suitable for a learning platform and an example of practical product architecture. In order to go further to improve on engineering, more detailed requirements need to be added, such as restrictions on the RF front end, restrictions on the scale of software and hardware, restrictions on the required supported frequencies and signals, etc. But by adding these details, the versatility of OPENGNSS as an architectural learning platform will be weakened. Moreover, OPENGNSS involves many functions such as support for broadband radio frequency, support for anti-interference, support for multi-system and multi-frequency, etc. If no trade-offs are made in the specific engineering implementation, its practicality will be weakened. Therefore, I decided to derive a new project specifically for consumer-grade GNSS receivers derived from Butterfly or its platform architecture, and further refine its requirements to receive navigation satellite signals centered on the L1 frequency point, including GPS L1C/A, GPS L1C, Galileo E1 and BDS B1C signals. An extension for L5 tracking engine is added afterward to support GPS L5, Galileo E5a and BDS B2a signals in the L5 frequency band to implement a dual frequency receiver. According to the further refined requirements, I significantly modified the C model and almost rewrote the entire baseband RTL logic. For example, for the same acquisition engine, the RTL implementation example given in OPENGNSS and the RTL implementation given in the Greta Oto project are two completely different implementations for different needs under the same architecture. This project is also an open source project, a complete baseband hardware C model, baseband hardware RTL, baseband control firmware and PVT software are all included as part of this project. A hardware platform that can actually run will also be added in the future if possible. Hope this platform will be continuously improved through feedback. Since this project is based on the Butterfly architecture and all designs are made transparent through open source, the name Greta Oto was chosen. I also hope that through this project, the aesthetic design will be presented to everyone, just like such a beautiful butterfly designed by the nature.

Copyright

1. Overview

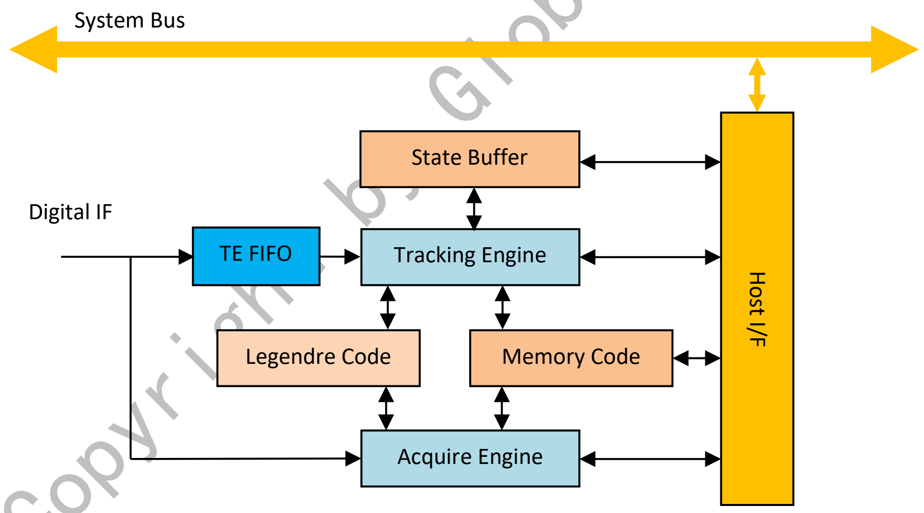
An overview of the design is introduced in this chapter.

1.1 Description of requirements

Requirements of the function and interface of the GNSS receiver are described in this section. The Greta Oto GNSS receiver is designed for consumer market, low cost and low power consumption, especially for IOT and similar products. It capable of tracking four signals centered at L1 frequency including GPS L1C/A (including SBAS), GPS L1C, Galileo E1 and BDS B1C. Because BPSK(1) or BOC(1,1) modulated signal requires 2MHz or 4MHz bandwidth respectively, the sample rate of the IF signal is designed to be near-zero complex signal with sampling rate higher than 4.092MHz. Each sample contains 4bit I and 4bit Q as sign/mag format. The MSB is sign bit (0 for positive, 1 for negative), the 3LSB are magnitude bits ranging from 000 to 111 represents 1,3,5,…,15. The single side bandwidth of RF front-end is 2MHz. If actual RF front-end parameters do not meet above requirements, a pre-processor is needed to adjust the signal.

1.2 Description of baseband hardware

The following figure gives the architecture of the baseband:



Two modules named tracking engine (TE) and acquisition engine (AE) acts as the signal processing units. They both accepts digital IF from RF front-end. The acquisition engine is used to do signal acquisition and do AE to TE aiding using synchronize signals between AE and TE. Tracking engine has four physical channels to implement dozens of logic channels by time-multiplexing. The TE is capable of tracking BPSK(1) and BOC(1,1) signals, with hardware NH code stripping and modulation data decode. The control and status register of all channels are mapped to local SRAM and accessible by CPU.

2. Design of tracking engine

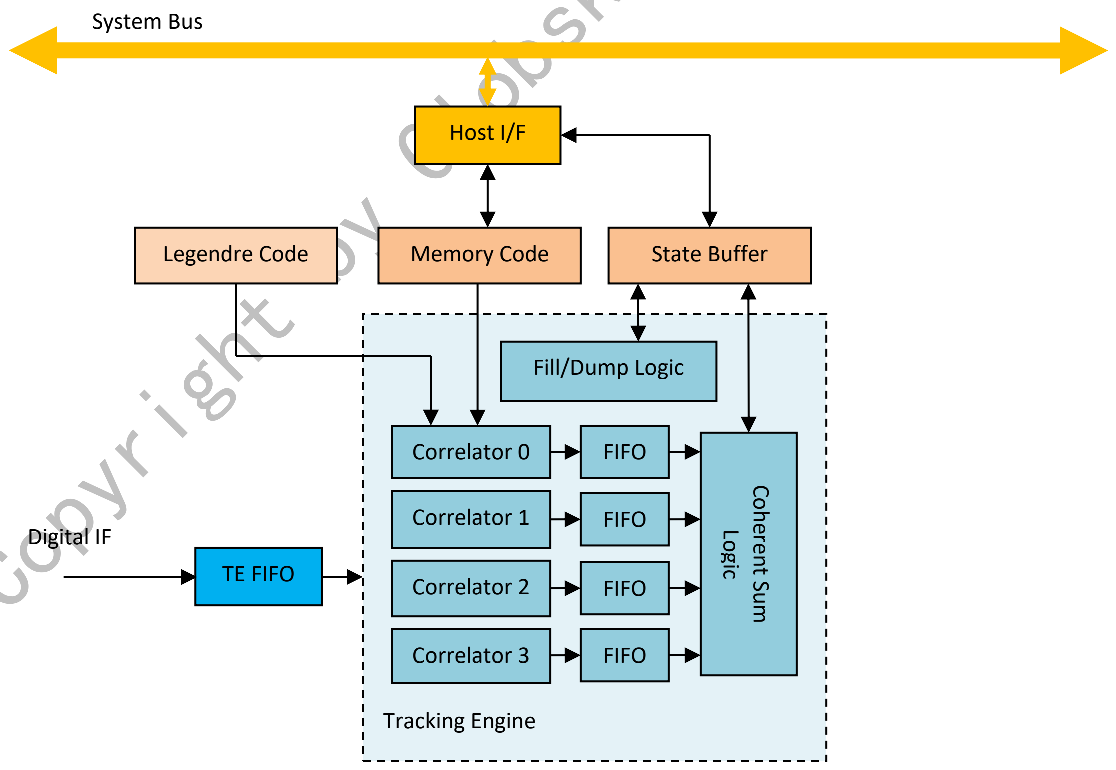
The tracking engine removes the IF frequency and residual of Doppler from the IF signal, and then do integration and accumulation after correlate with different PRN code. The IF samples are saved into the sample FIFO. The multiplexing of correlator is implemented by read sample from FIFO repeatedly.

The TE hardware works together with the software alternatively. After the TE hardware completes the processing of 1ms samples, if correlation results or measurement data are available, an interrupt is generated and the TE will halt before CPU resumes it. CPU can access the control and status register of any channel, read correlation results, do tracking loop and add/delete channels within ISR and resume TE before returning from ISR.

The TE is designed to process one sample in each clock cycle. So each physical channel requires 4 million clock cycles to process IF data at 4MHz sampling rate. In order to have 48 logical channels (GPS, Galileo and BDS each occupies 16 channels), a total 192M clock cycles is required, which means 48 million cycles when there are four physical channels. So hardware will occupies half of the time if it runs at 100MHz and gives enough time for software access.

2.1 Brief description of TE

The figure of the TE architecture is shown below:



TE has following function modules: Correlator 0~3 represents 4 physical correlation channels to correlate and integrate IF data. Each physical channel contains 8 correlators to get correlation results of different delay. Each physical channel is followed by a correlation result FIFO sending data to coherent sum module. This module is used to calculate coherent sum of correlation results through multiple milliseconds. A fill/dump module is used to fill channel register contents from the SRAM called TE state buffer and dump contents back to the TE state buffer during physical channel multiplexing. All contents of the channel registers are mapped to the SRAM named state buffer. Which means from CPU side of view, read/write state buffer is equivalent to access to channel registers (when TE is halted). The subsequent of write state buffer during TE is running is unpredictable and need to be avoided. Other function modules include TE FIFO used to buffer IF data, RAM/ROM used to generate Galileo memory code and L1C/B1C Weil code etc.

The time sequence of TE processing is:

Step 1: Wait samples buffered in TE FIFO reached at least 1ms (this 1ms samples are called data block.

Step 2: TE FIFO signals the fill/dump module.

Step 3: The fill/dump module fills register contents of active channel to up to 4 physical channels.

Step 4: Correlator signals the TE FIFO to output current data block and do correlation.

Step 5: If any channel reaches 1ms boundary of PRN code during correlation, correlation result is output to correlation result FIFO.

Step 6: The fill/dump module dumps register contents of active physical channel back to state buffer.

Step 7: Repeat step 3~6 until all active logic channels completed.

Step 8: TE FIFO discard current data block.

Step 9: Interrupt CPU if needed and wait CPU to resume TE.

Step 10: Go back to step 1.

The following sections will give detailed description of each module in TE one by one.

2.2 TE FIFO

The purpose of TE FIFO is to buffer input IF samples, then inform the correlator to start processing data after a certain amount of samples are buffered (normally 1 millisecond, called one data block). The principle of TE FIFO is to maintain a read pointer and a write pointer. Write pointer will continuously increase when new IF samples come. Read pointer will go through the current data block repeatedly until all channels completed, then skip to the start of next data block. A sample number counter is maintained to implement data overflow protection.

Because the correlation channel operates using time multiplexing, so the time sequence is controlled by both software and hardware. Because the write pointer register is the only register acts as sample level counter maintaining local time, it is treated as clock source runs at nominal

IF sampling frequency. The write pointer register is also used to synchronize with other events (refer to section of write pointer latch).

In order to implement the functions of TE FIFO, following features and functions are designed:

2.2.1 Reset/Enable/Clear

Reset signal will force all registers in TE FIFO to be assigned to initial value. Both asynchronous reset (on hardware startup) and synchronous reset (controlled by software) are supported. Enable signal is used to control whether TE FIFO is active. If TE FIFO is disabled, read/write operation is not performed. The TE FIFO enable signal can be controlled by software or synchronous event.

Clear signal is used to set all registers to initial value except control registers.

2.2.2 Dummy Write

Dummy write is mainly used to implement duty cycle. When invalid IF samples feed to FIFO, the dummy write is enabled by software. At this state, only write pointer and sample count register is changed; write operation on TE FIFO SRAM is muted. So during duty cycle mode, the whole RF front-end can be periodically disabled except sampling clock to maintain local time.

2.2.3 Overflow protection and threshold

The TE FIFO can only buffer certain amount of sample data which depends on the size of the TE FIFO RAM. When the data buffer exceeds the SRAM size, overflow occurs and an overflow flag is set. The overflow flag can only be cleared by resetting TE FIFO or by CPU writing clear command. The TE FIFO will hold one millisecond data when it informs the correlation channel to start, then data continuously accumulate until all channels completed. In the worst case, the correlation channel will need up to 1 millisecond to complete, so the TE FIFO need to hold at least 2 milliseconds of data. Considering there will be extra time for CPU to do data read/write during ISR, TE FIFO needs a little more spaces. For example, at 4MHz sampling rate and totally 8bit I/Q data, a RAM with size 10kB is suitable for TE FIFO to hold 2.5ms data.

When overflow occurs, new input sample data will overwrite the current data block in use. In this case, incorrect correlation result will be generated. On the other hand, because the sample count register is designed to have much larger range than the size of TE FIFO RAM, the overflow will only give incorrect sample data to correlation channel, the read/write register and sample count register remains correct value. So the TE will recover at the next data block. The correlation result will not be much affected if the overwritten samples are not too much. This method protects the stability of TE.

A threshold exceed indicator is used as another method for protection. This flag is a read only bit in status register, set when sample count exceeds threshold and clear otherwise. CPU can use this indicator to determine whether immediate return from ISR and resume TE is needed to prevent overflow.

2.2.4 Rewind/Skip

Based on the time sequence of TE processing, all correlation channel process the same data block, so TE FIFO will output the samples in current data block repeatedly and then discard the current data block. This is implemented by applying two operations: rewind and skip. After complete output the current data block, rewind operation forces the read pointer goes back to the originally position to output current data block again in the next round, while skip operation forces read pointer to add the value of block size to discard the current data block. The sample count register will also decrease by the value of block size on skip operation.

2.2.5 Write pointer latch

Because the write pointer reflects the highest resolution part of receiver local time, so it is important to have this time to synchronous to other events by latching the value of write pointer at certain time.

The write pointer is designed to be latched to four individual registers on four different events. The first event is CPU command; the second event is external event mark; the third event is PPS pulse; the fourth event is AE event (usually occurs when the first sample written into AE buffer). There are three sections of latched value: round number, write pointer and write sub count respectively. The write pointer can only represent the time span of the size of TE FIFO (2.5ms in previous example), so a round number register is also latch. The round number register increase each time the write pointer goes back to the beginning of TE FIFO, which gives larger range in time. A sub count register will increase on each system clock and reset to 0 when write pointers increases. The use of this register increases the resolution of latch time from the interval of sample clock up to the interval of system clock.

2.2.6 Block size adjust

This is an optional feature and not usually used. Generally, the data block size is fixed to nominal sample numbers within 1 millisecond. The register values of code phase, code count and carrier phase of each channel reflects the values of the signal at the epoch of last sample of current data block when the TE completed.

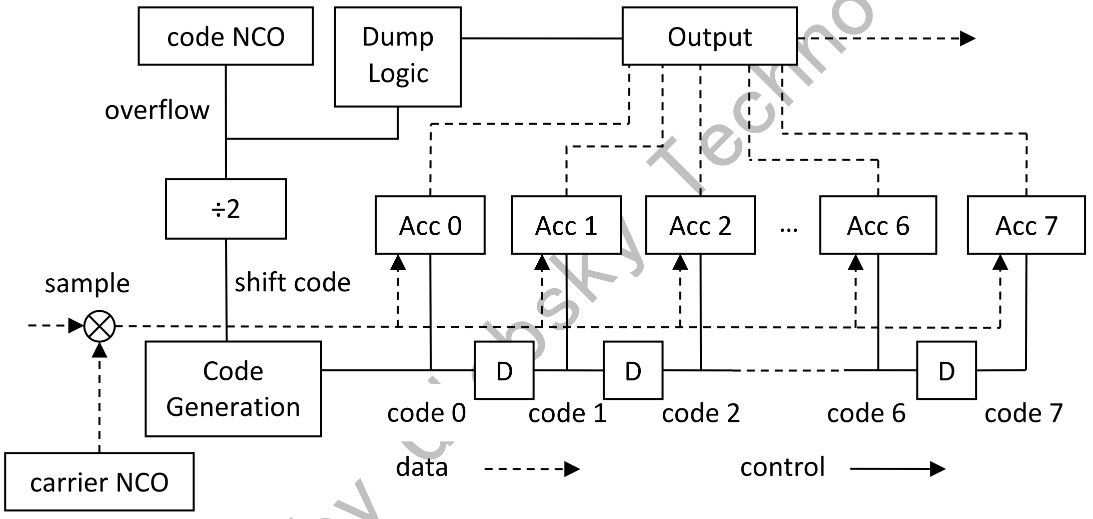
In most cases, the block size does not need to be adjusted. If the clock error is larger than a certain threshold, observation epoch can be adjusted forward or backward 1 millisecond (a whole data block) by software. But in some special user scenario, the observation time need to be as close to second edge as possible. In this case, the size of data block need to be changed, so higher resolution of observation time is possible.

The adjustment of block size is in unit of sample, with range from -128 to 127. The adjustment will only effect once when software writes the adjustment value and cleared by hardware automatically after change.

2.3 Correlation channel

The main function of the correlation channel is to do correlation and integration with the IF data in the TE FIFO after stripping the residual Doppler (and nominal IF frequency) and the local codes with different delays, then output the correlation result at the edge of integration period. Since the code cycles of the local codes with different delays end at different times, the outputs of correlators with different delay will be output in sequence due to their different code edge. The correlation cycle can be the same to the code cycle, or it can be shorter than one code cycle but align to the code edge. For example, in the case of Galileo, one code cycle is 4092 codes, and the correlation cycle can be set to 1023 codes, which is 1 millisecond. The cases of L1C and B1C are similar. However, regardless of whether the length of the code and integration period is the same, the edge of should be aligned.

The functional block and data flow chart of correlation channel is shown below:



The input sample is first mixed with local carrier. The local carrier is generated by a 32bit NCO controlled by a 32bit carrier frequency control word (FCW). The 6MSB of carrier NCO is used as index to a sin/cos lookup table. The amplitude of sin/cos value is 12.

The mixed signal is then correlated with local PRN code. The generation of PRN code is controlled by a code NCO which generates a code overflow signal at the frequency of twice of code rate. The overflow signal is used to do code shift between local code registers of different delay, and also used to control code generation module after its frequency divided by 2.

The modules marked Accx are accumulation module, one for each correlator. It accumulates the correlation result of input signal and corresponding local PRN code. The accumulation result is a complex number with 16bit for both I and Q. The result will output at the edge of integration period and start from 0 to continue integration.

2.3.1 Code Jump

The tracking loop normally targeted to fix the correlation peak at certain correlator (eg. correlator 4 in within eight correlators) by code FCW. But in the cases of acquisition to tracking process or during TE acquisition, it is required to shift local code forward or backward relative to input signal for a number of correlator intervals. The implementation method in correlation channel to achieve such code shift is called code jump. The code jump is controlled by a 8bit register which contains a 2’s complement number. Jump count of 0 means there is no shift; jump count with a positive value will force code NCO overflow corresponding times before process any IF data; jump count with a negative value will force to mute the code NCO overflow signal for corresponding times during correlation process. This method will make correlation peak move forward (to the larger index correlator) if the jump count is positive and vice versa.

2.3.2 Narrow correlator

The code NCO is designed to overflow at the frequency of 2 times of code rate, which results the correlator interval to be fixed 1/2 chip. In some cases like better resolution of delay discrimination or better multipath mitigation result, narrow correlator is needed, especially for BOC signal tracking which has much narrower correlation peak. Narrow correlator implementation method is described below:

When code NCO is configured to overflow at the frequency of twice of code frequency, the code NCO FCW will be calculated by $\mathrm{F_c} = \frac{2\mathrm{f_c}}{\mathrm{f_s}} 2^{32}$, in which $\mathrm{f_s}$ is sample frequency, $\mathrm{f_c}$ is code frequency which is 1.023MHz for all signals centered at L1.

So the code phase of any correlator n can be expressed as:

$$\Phi_\mathrm{n}(i) = [\mathrm{NCO}(i) - \mathrm{d_n} \cdot 2^{32}]/2^{32}$$

In the above equation, $\Phi_\mathrm{n}(i)$ is the code phases for sample i within data block, NCO(i) is the value sequence in code NCO register, $\mathrm{d_n}$ is the interval between correlator n and peak correlator. The integer part of $\Phi_\mathrm{n}(i)$ is the local PRN code index for correlator n.

When $\mathrm{d_n}$=1, it means correlator n need to wait overflow signal extra one time to reach the same code phase as peak correlator, or 1/2 chip delay equivalently. It is the same for $\mathrm{d_n}$=-1 which means 1/2 chip advance. Generally, $\mathrm{d_n}$ is a integer value for all correlators have 1/2 chip interval between each other.

If we choose $\mathrm{d_n}$=1/2, then $\Phi_\mathrm{n}(i)$ will be the code phase sequence half of the correlator interval delay to peak correlator or 1/4 chip delay to peak correlator. In this case, we will have $\Phi_{1/2}(i) = \Phi_0(i) - 2^{31}$, which means if fractional part of $\Phi_0(i)$ is larger than 2^{31}, $\Phi_{1/2}(i)$ has the same integer part as $\Phi_0(i)$, so current correlator uses the same PRN code as peak correlator. Otherwise, integer part of $\Phi_{1/2}(i)$ is one less than the integer part as $\Phi_0(i)$, so current correlator uses the same PRN code as the code after 1/2 chip delay.

For d_n=-1/2, we will have similar conclusion. So in the original design, we have peak code sequence, 1/2 chip advance code sequence and 1/2 chip delay code sequence, the 1/4 chip advance and 1/4 chip delay code sequence can be derived with following table:

Code sequence	MSB of NCO	Code
1/4 chip advance	0	Prompt code
	1	Advanced code
1/4 chip delay	1	Prompt code
	0	Delay code

Similarly, for d_n=1/4 or d_n=-1/4, 1/8 chip delay code and advanced code can be derived using 2MSB of code NCO:

Code sequence	2MSB of NCO	Code
1/8 chip advance	others	Prompt code
	11	Advanced code
1/8 chip delay	others	Prompt code
	00	Delay code

For the case of correlator 4 set as peak correlator, correlator 3 and 5 can be chosen to use 1/4 chip advanced and delay code respectively and correlator 2 and 6 to use 1/2 chip advanced code and delay code, then we get 5 correlation result with 1/4 chip interval between each other. With the same method, 1/8 chip interval correlation result can also be calculated.

2.3.3 BOC PRN

All signals modulated at L1 frequency is BOC(1,1) modulated except for L1C/A signal. The BOC(1,1) signal can be tracked either using single sideband or the dual-band signal. For single sideband tracking, it can be treated as a normal BPSK(1) signal with 1.023MHz carrier shift. For dual-band tracking, the code generator will generate new PRN code on every other overflow signal. So for the first overflow signal, the PRN code output from code generator will go into code delay shift registers; for the second overflow signal, the opposite value will go into code delay shift registers.

It needs to be noted that the correlation peak is much narrower for BOC signal than for the BPSK signal, so 1/8 chip correlator interval is suggested to do BOC tracking.

2.3.4 Data channel and pilot channel

All signals modulated at L1 frequency has both data channel and pilot channel signal except for L1C/A signal. The pilot channel signal is usually used for tracking not only because it usually has higher power, we can also use longer coherent integration time for pilot signal.

The demand for doing data decode when tracking the pilot channel results the option to use alternative code for correlator 0. In this case, correlator 1~7 use the code in code delay shift registers comes from primary code generator as usual, and correlator 0 uses the code generated from secondary code generator with 2 code delay (to be aligned to correlator 4 as peak correlator).

2.3.5 NH code stripping

All signals modulated at L1 frequency has NH code modulated except for L1C/A signal. In order to have hardware implement coherent sum by itself, NH code must be stripped from correlation result. This is implemented by a 25bit NH code register, a 5bit NH length register and a 5bit NH count register as NH current index within NH sequence.

If NH code stripping is enabled, the current NH code is XOR to the PRN code. After each code round, the NH count will increment by 1 until NH length reached, then go back to 0.

For Galileo E1C, NH length can be set to 25 and the NH code will be stripped automatically by hardware after the correct NH count is set. But for L1Cand B1C, which has 1800bit NH code, software need to cooperate to do NH code stripping. The 1800bit NH code can be divided into several segments and use software to write new NH sequence after current segment ends. It should be noted that data block edge does not align to code edge, so N data blocks will cross N+1 milliseconds of code, which means 250 data blocks contains 26bit NH code in the worst case. So for L1C and B1C, the suggested segment length is 20bit instead of 25bit, and 90 segments are used. Each segment is actually at least 21bit and has at least 1bit overlap between each other. When NH count is equal or greater than 20, software need to write NH code of next segment.

2.3.6 Correlation result output and coherent sum

Each correlator will output the integration result and clear the accumulator at the end of every integration period. Because the integration period is aligned to the code of different delay, so the correlation result of each correlator will be output sequentially. Each physical tracking channel has its own correlation result FIFO. The coherent sum module will scan each correlation result FIFO cyclic and add the correlation result into coherent sum buffer (occupies part of TE state buffer).

Each correlation result in FIFO has 32bit result data (16bit I and 16bit Q), a logic channel index, a correlator index, and two indicator bits. A depth of 8 for each FIFO is enough for each correlation channel to prevent correlation result lost.

The first indicator is start of coherent sum, which means current result is the first correlation result within coherent sum period or subsequent result. This bit controls whether clear coherent sum result before adding current correlation result. The second indicator is overwrite protection indicator which is described below.

Generally, because the length of data block and the integration result are both 1 millisecond, each correlator will output correlation result once during each data block processing. But it is possible that one correlator will output correlation result twice in one data block in three cases. The first case is code jump has positive value; the second case is positive code Doppler, the third case is positive block size adjust value. In these three cases, it is possible for one or more correlator to output correlation result both at the beginning of a data block and at the end of data block. The second case will definitely happen during normal tracking, so special operation is needed to deal with such possibility. The correlation channel will save the index of correlator when it first outputs the correlation result within one data block. If the subsequent correlation result comes from the same correlator, the overflow protection indicator is set.

If the first result is the last integration period within coherent sum period, the second result will overwrite the stored coherent sum result before the previous one read out by CPU.

The coherent sum module will determine whether the correlation result has both indicators set. In this case, the second correlation result will not be put in coherent sum buffer to prevent the previous coherent result been overwritten. Instead, the second correlation result and corresponding channel and correlator index will be put into a set of registers for software. Because the possibility of overflow protection is relatively low, the case of more than one channel to have overwrite protection in the same data block is ignored, so there is only one set of such registers.

Software will check the valid flag of overwrite protection, and performs corresponding operation for such case. Document for firmware design will describe it in detail.

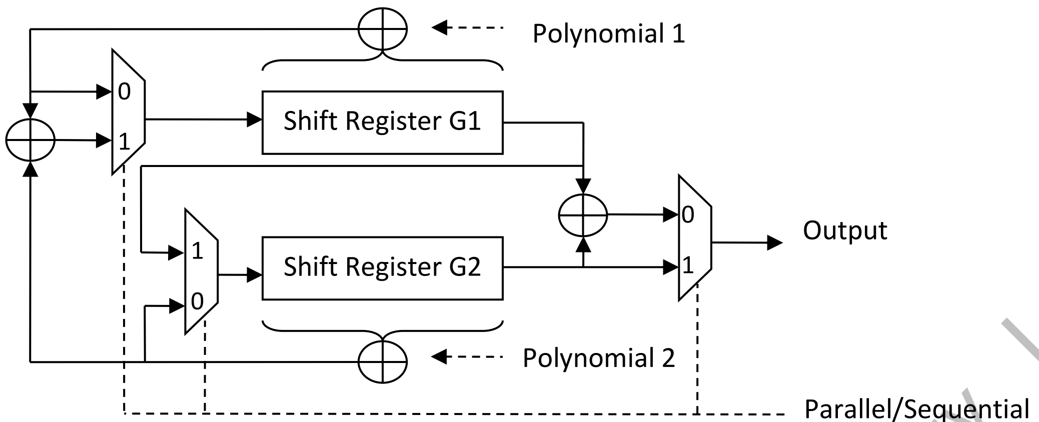
2.4 Code generator

The code generator is part of the correlation channel. The tracking engine is adaptive to track signal with different PRN type by having correlation channel use different code generator. In order to support all signals in both L1 and L5, three different code generators are implemented: The first one is generic LSFR register used to generate Gold code for GPS L1C/A, GPS L5, Galileo E5a and BDS B2a; the second one is Weil code generator for GPS L1C and BDS B1C; the third one is memory code generator for Galileo E1.

These three code generators have the same or similar hardware interface to configure registers, read registers, and generate PRN code. The detailed design of these code generators are described below.

2.4.1 Generic LSFR

Two generic left shift feedback registers (LSFR) composes a Gold code generator as following:



Each LSFR is 14bit in length. Each register has configurable registers to set initial phase and generation polynomial, and a code length register to control the reloading of initial phase at the end of each code period. The outputs of two LSFRs are XOR to generate Gold code. 28 bit sequential LSFR mode is also supported to have it reuse in other project.

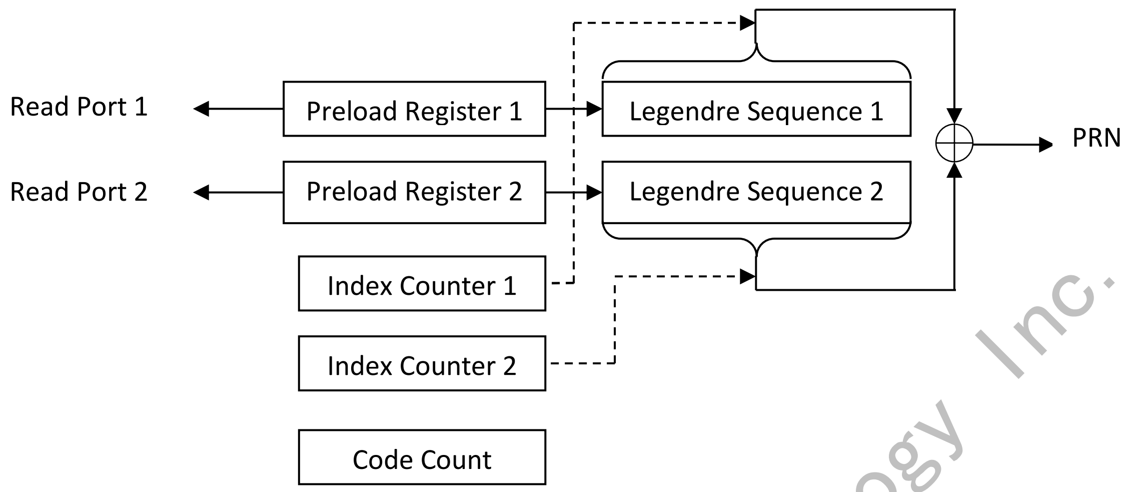
Generation polynomial and code length are universal for all signals in L1 (GPS L1C/A and SBAS), initial phase (as configuration), current phase and PRN count (as status) are different for each correlation channel.

2.4.2 Weil code generator

The GPS L1C and BDS B1C has similar Weil code generation method, so they can be generate use the same module and uses input signal weil_type to select different Weil code. The Legendre code to generate Weil code is stored in a ROM. There is only one GPS Legendre code ROM and one BDS Legendre code ROM. Two Legendre code generators are used in Weil code generator; each has its own 16bit parallel read port to access Legendre ROM. For each correlation channel, data code generator and pilot code generator totally has four read port. These four port access the same Legendre ROM through an arbiter. In this project, because there are four correlation channels, a second level arbiter is used to achieve Legendre ROM multiplexing.

For the case of sampling rate at 4 times of code rate, a new Weil code is needed every 4 clock cycles and each Legendre code generator need to access the Legendre ROM every 64 clock cycle through 16bit interface. So there is enough bandwidth for 16 Legendre code generators to read from one Legendre ROM.

The figure of Weil code generator is shown below:

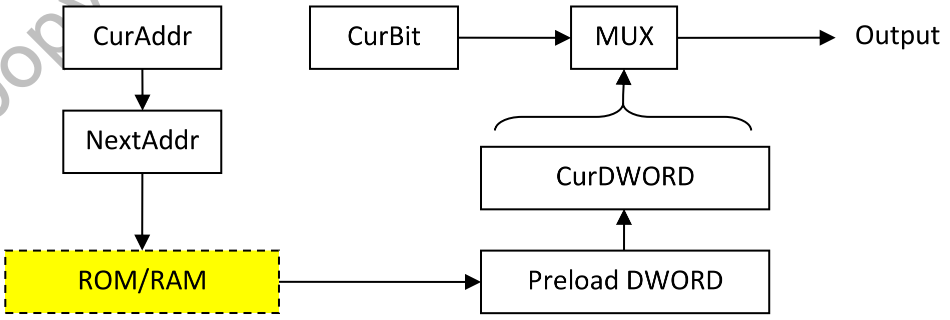


Index counter 1 and index counter 2 counts the current index in Legendre sequence, and code count is the current code index in Weil code. Legendre sequence 1 and 2 are two 16bit registers. The output of these two registers selects the bit using 4LSB of corresponding index counter. After the last bit output, the Legendre sequence register filled with the value in preload register to ensure continuous output. The preload register then loads the next 16bit in Legendre sequence from Legendre ROM.

The configurations for different SVs are Weil index and insertion index (L1C) or truncation point (B1C). The status for different SVs is Legendre sequence and code count. Contents of other registers including index counter and preload register will be calculated or filled automatically after fill operation during channel context switch.

2.4.3 Memory code generator

Memory code generator is used to generate Galileo E1 PRN code (both E1B and E1C) by reading pre-stored PRN code in ROM or RAM. Because the code length is multiple of 1023, so the data width of ROM is 32bit DWORD. Each segment in ROM contains 32DWORD or 1023bit (by ignoring LSB of last DWORD). The logic of memory code generator is shown below:



The Current DWORD register and preload DWORD register storing memory code sequence are both 32bit. Current bit register is a 5bit register used to select the output bit in current DWORD register. Preload DWORD register ensures the continuous output of memory code.

The configure register selects different SVs by setting the start segment index in memory code ROM. For Galileo E1, each satellite needs 4 segments to store E1B and 4 segments to store E1C which occupies 256 DWORD, so a ROM with size of 36kB can hold memory code for all SVs.

A two port arbiter is used for E1B and E1C generate at the same time and a four port arbiter is used for four correlation channels run at the same time.

2.5 Channel state switch

The core function of channel multiplexing is the context switch of channel state, which is performed by fill/dump module. A global register will logic channel is active within all channels. On starting the correlation channel, the fill/dump module will first put all contents in state buffer to the registers of physical channel. After completing the process of one data block, the contents of status registers and counter registers will be dumped and write back into state buffer (contents in configuration registers will not be changed and do not need to write back).

The contents of all channel registers are combined into 32bit DWRODs to map to different address in state buffer. Each logic channel occupies 32DWORDs in state buffer: first 16 DWORDs for configuration and status register; next 8 DWORDs hold current integration result of 8 correlators; the last 8 DWORDs occupied by coherent sum result. The coherent results were accessed by coherent sum module. They are not part of the context of a logic channel but put in the same SRAM.

The placement of registers of one logic channel mapped in state buffer is shown below:

	Alias	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	CARRIER_FREQ	CarrierFreq																															
1	CODE_FREQ	CodeFreq																															
2	CORR_CONFIG	Reserved					Coh Number						BitLength					Reserved				(7)		(6)		(5)	(4)	(3)	R	(2)		(1)	
3	NH_CONFIG	NH Length					R		NH Code																								
4	COH_CONFIG	Reserved																DumpLength															
5	PRN_CONFIG	Buffer of Code Generator																															
6	PRN_STATE																																
7	PRN_COUNT																																
8	CARRIER_PHASE	CarrierPhase																															
9	CARRIER_COUNT	CarrierCount																															
10	CODE_PHASE	CodePhase																															
11	PRN_CODE	DumpCount																JumpCount								PrnCode							C0
12	CORR_STATE	NH Count					Coherent Count						BitCount					PrnCode2				R			(11)	(10)	CurrCor			R		(9)	(8)
13	MS_DATA	DecodeData																															
14	PRN_CONFIG2	Buffer of Second Code Generator																															
15	PRN_STATE2																																
16		Partial Accumulator I of Correlator 0																Partial Accumulator Q of Correlator 0															
17		Partial Accumulator I of Correlator 1																Partial Accumulator Q of Correlator 1															
…		…																…															
23		Partial Accumulator I of Correlator 7																Partial Accumulator Q of Correlator 7															
24		Dumped Accumulator I of Correlator 0																Dumped Accumulator Q of Correlator 0															
25		Dumped Accumulator I of Correlator 1																Dumped Accumulator Q of Correlator 1															
…		…																…															
31		Dumped Accumulator I of Correlator 7																Dumped Accumulator Q of Correlator 7															

(1) Pre Shift	(2) Post Shift	(3) Data in Q Branch
(4) Enable Second Code Generator	(5) Enable BOC	(6) DecodeBit
(7) Narrow Factor	(8) Coherent Done	(9) Overwrite Protect
(10) Dumping	(11) Code Subphase	

Legend:

🟦 Configuration Field

🟩 Status Field

The mappings of code generators are different depend on the type of code generator, which are distinguished by 2MSB of the PRN_CONFIG field:

General PRN Generator

	Alias	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
5/14	PRN_CONFIG	0	0	R		G2 InitState														G1 InitState													
6/15	PRN_STATE	R				G2 CurState														G1 CurState													
7	PRN_COUNT	GlobalCount																		G1 CurCount													

Weil Code

	Alias	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
5/14	PRN_CONFIG	1	0	(1)	R	Insertion Index														Weil Index													
6/15	PRN_STATE	Legendre Code 2																Legendre Code 1															
7	PRN_COUNT	Reserved																		Current Count													

Memory Code

	Alias	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
5/14	PRN_CONFIG	1	1	Reserved														Start Index												Length			
6/15	PRN_STATE	Current Code																															
7	PRN_COUNT	Reserved																		Current Count													

(1) Weil Code Selection

Note: 4MSB of PRN_CONFIG2 is ignored, use 4MSB of PRN_CONFIG instead

Global polynomial registers

	For parallel																																
	Polynomial Set Reg0	0	R			G2 Polynomial														G1 Polynomial													
	Polynomial Set Reg1	GlobalLength																		G1 Length													
	For sequencial																																
	Polynomial Set Reg0	1	R			G2 Polynomial														G1 Polynomial													
	Polynomial Set Reg1	GlobalLength																															

2.6 Noise floor calculation

The noise floor calculation module is used to calculate the average or smoothed noise in input IF signal. The noise floor will be used for two purposes: the first one is to divide the signal power to calculate the signal SNR or equivalently the C/N0; the second one is to monitor the average power of input IF, which gives another way of AGC control to RF front-end.

In order to get the noise power more accurate, the noise floor calculation module calculates the average power of signal within 2MHz bandwidth centered at L1. When processing data of first active logic channel (which will be put in physical channel 0 for the first round), the signals with Doppler removed will first correlate with a 1023-length m sequence generated by clock at code rate and then integrate for a whole period of m sequence. The integration result is considered as the noise power normalized to 1 millisecond.

A simplified method is adopted by using amplitude of the 1 millisecond result instead of the power to avoid multiplication. The amplitude is calculated using JPL equation, and then use equation $N_{n+1} = N_n + k(A - N_n)$ to calculate the smoothed value iteratively. Here k is the smooth factor ranging from 1/256 to 1/16384. The smaller then smooth factor, the noise floor is more stable.

It is assumed that the noise is wideband Gauss white noise and the distribution of 1 millisecond integration result follows independent normal distribution for both I and Q. If the standard deviation of 1 millisecond integration result is σ for both I and Q, the amplitude follows Rayleigh distribution with mathematical expectation at $\sigma\sqrt{\frac{\pi}{2}} \approx 1.2533\sigma$. Which means when you get the smoothed amplitude, σ can be calculated and normalized 1 millisecond power $2\sigma^2$ is get.

2.7 Hardware data decode

For signals with both data and pilot channel, correlator 1~7 are used for correlation on pilot channel while correlator 0 used for data channel. The coherent result calculated from correlator 0 can be used to demodulate the navigation data. During the stable tracking stage, the coherent time for pilot signal is typically 20 milliseconds or longer, which is larger than the length of data bit (for E1 is 4ms, for L1C and B1C is 10ms). Because will not be interrupted during the middle of coherent sum period, the coherent result of correlator 0 will cross multiple data bit and cannot be decoded unless the decoding is done by hardware.

After the phase of pilot channel is locked, the data modulated on data channel can be calculated using I or Q part of integration result of correlator 0 (depends on whether data is modulated in-phase or quad-phase). The decoding of modulation data can use either hard decision or soft decision. Hard decision uses the sign bit of the coherent sum result, soft decision scale the coherent sum result to 4bit or 8bit. 4bit soft decision is suggested to be used for Viterbi decode and 8bit is suggested to be used for LDPC decode. Soft decision is implemented by choosing 4 or 8 bits from the 16bit coherent sum result with overflow protection. The bit selection method needs to retain as much effective bits as possible on different signal strength.

The coherent length of data channel will not be longer then the pilot channel, and the signal power of data channel is also equal or less than that of pilot channel. So if the pilot channel coherent sum result does not overflow, the data channel result will not overflow either.

The following calculation gives the guide of bit selection for soft decision:

First the highest data signal SNR is considered to be at 16dB (equivalently 46dBHz, or 49dBHz total power for E1 and 52dBHz total power for L1C/B1C). Signals with higher power will saturate, equivalent to hard decision, and will not increase BER for such a strong signal.

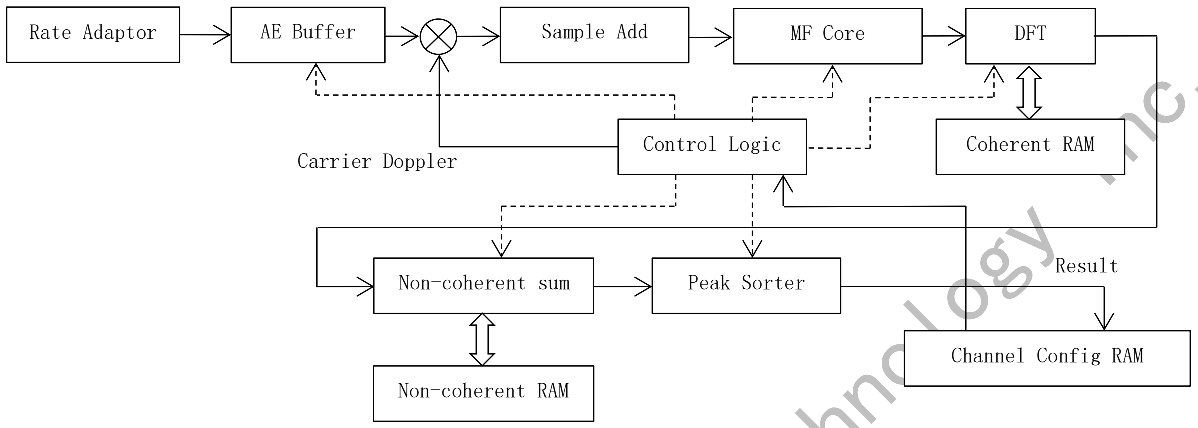
For 4bit sign/mag optimized quantization, the standard deviation is about 625 for 1 millisecond I or Q integration result at 4MHz sampling frequency. Put $\sigma = 625$ and SNR=16 into the equation $\mathrm{SNR} = 10 \cdot \log_{10} \frac{A^2}{2\sigma^2}$, we will get signal amplitude A as 5576. So the amplitude of E1 with 4ms coherent period is 22304, the amplitude of L1C or B1C with 10ms coherent period is 55760. They can be represented using 16bit and 17bit respectively.

The value will be smaller if pre-shift and post-shift bit not equal to 0. So a bit selection factor S is calculated with $S = B_{\mathrm{pre}} + B_{\mathrm{post}} - L$. Here L equals 0 for E1 signal and equals 1 for L1C and B1C signal. It is obvious that when S equals 0, the coherent sum result can be represented using 16bit; when S equals 1, the coherent sum result can be represented using 15bit etc.

The soft decision bit selection uses the value of S to automatically adapt to different configuration. When S equals 0, the bits are selected starting from MSB (bit15) of the coherent result; when S equals 1, the bits are selected start of bit14 etc. If the omitted MSBs do not equal to the sign bit of the selected bits, saturation is performed.

3. Design of acquisition engine

The following figure is the architecture of acquisition engine which is capable to do acquisition on GPS L1C/A, GPS L1C, BDS B1C and Galileo E1 signals.



The input IF signal is first down-sampled to two times of code frequency after removing nominal IF, then stored into AE buffer. The acquisition engine can only acquire BPSK like signal, so BOC(1,1) signal need to have extra 1.023MHz IF removed.

The core function module of acquisition engine is the match filter. It calculate the dot product result on input sample sequence and PRN code sequence using adder tree, and then shift the input sample to calculate the correlation result of different code phase.

The output of match filter will first multiply with four different rotate factors to calculate DFT results on 8 frequencies. This will have the acquisition engine to search 8 frequencies in parallel and speed up the acquisition. The intermediate result of DFT is expressed as 10bit I plus 10bit Q plus a 4bit exponent format (named as exp10 format) and stored into coherent RAM. After coherent sum completes, amplitude will be calculated from coherent result. Then the amplitude will be added together to get non-coherent result. The non-coherent result be stored in non-coherent RAM using block float format, which means all results use the same exponent.

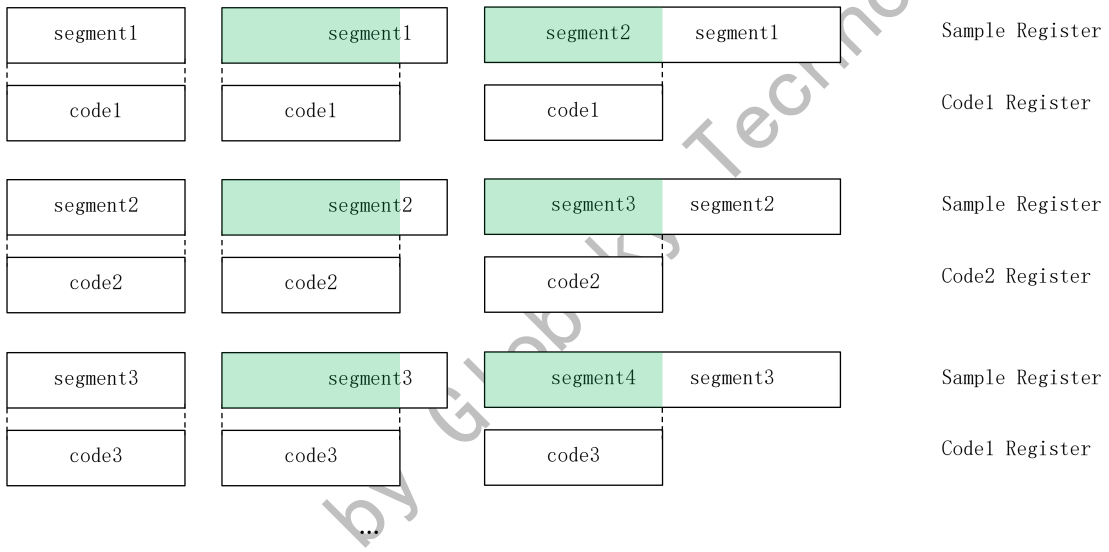
Finally, non-coherent result will be compared in peak sorter to get three largest amplitude results and corresponding frequencies and code phases.

3.1 Timing sequence of AE

The sampling rate of signal used in AE is twice of code frequency, so the corresponding code phase resolution of acquisition result is 1/2 chip. Depend on the different code length of different signals to be acquired, the total code phases need to be searched for GPS L1C/A, Galileo E1 and GPS L1C/BDS B1C are 2046, 8184 and 20460 respectively.

To minimize the number of logic gates and RAM size used in AE while having relatively fast speed of acquisition, the depth of match filter is designed to be 682, which is one third of 2046. This means we need to add 3 of match filter output with the same code phase together to get the correlation result of 1 millisecond. This also means the number of code phases to be search within one round is 682. The search process need to be performed for multiple rounds to have all code phases been scanned.

The match filter contains one set of shift registers with 12bit width (6bit I and 6bit Q) and 682 depth to hold the input sample, two sets of shift registers with 1bit width and 341 depth to hold the code. The two sets of code shift registers are used alternately. For each round of 682 code phases search, the timing of operation is given in the following figure. At first, in preload stage, the sample shift register and first set of code shift register are filled with samples and PRN code of first 1/3 millisecond (e.g. 341 PRN code and 682 samples named as segment). With the match filter calculate and output the result of current code phase, new sample shift in and the correlation result of next code phase calculated one by one until the 682th result been calculated. At this time, the first segment data shift out and the second segment data shift into the sample shift registers as the first row of figure shows. Then the first round of correlation completed.



During this period of time, the second set of code shift register filled with next 341 PRN code. Then the code shift register used by match filter switch to second set. Because the data segment 2 is aligned with code segment 2 at this time, the match filter begin to output the correlation results of next 1/3 millisecond of the same 682 code phases in the first round as the second row of the figure shows. During the operation of second round, the first set of code shift register is then loaded with next 1/3 millisecond PRN code. So the third round of operation performed as shown in row 3 of the figure.

Adding match filter results of the same code phase on three consequent segments to get 1 millisecond result, applying rotation factor and calculate coherent sum, adding amplitude to get non-coherent results, sorting the non-coherent result to get the largest three are performed pipe lined. So the match filter works continuously until the search of first 682 code phases completed. Then the above process start over again with the code preload from beginning but the sample load starting from data segment 2, this will start the search of next 682 code phases.

The AE hardware control logic will do 7 nested loops to complete the search tasks. The most inner loop is to shift sample register 682 times to have the match filter calculate the 1/3 millisecond correlation results of 682 code phases. The second loop is to add the result 3 times to get correlation result of 1 millisecond. The third loop is to rotate the 1 millisecond result and do coherent accumulation on 8 frequency bins in parallel. The fourth loop is doing non-coherent sum on the amplitude of coherent sum result. The fifth loop is to scan multiple 682 code phases until all desired code phases are searched. The sixth loop is to search on different Doppler because the search bandwidth is limited depending on the length of coherent sum time. The seventh loop is to search different PRN code of different SVs.

The AE will complete all above 7 nested loops to search multiple SVs without intervention from CPU. So after CPU set all search configures and starts the AE, it will wait for the AE interrupt to get the final search result.

3.2 Rate adaptor module

The rate adaptor module processes the input IF signal in three stages before putting the result into AE buffer. The first stage is to remove the nominal IF (including 1.023MHz offset if BOC signal need to be acquired). The second stage is to filter and resample the signal to twice of code frequency, e.g. 2.046MHz. The third stage is to do re-quantization on the resampled signal to 2bit sign/mag format for both I and Q.

The carrier NCO uses the same one as adopted in TE. After a filter with 1MHz single side band, the signal is down-sampled to 2.046MHz using a 24bit NCO. Because the sampling frequency of input signal is assumed to be a little higher than 4MHz, which is considered to be 4 times of the filter bandwidth, so a simplified design is adopted by using constant value as filter coefficients. The anti-alias filter is a 6th order FIR filter with coefficients as -5, 8, 20, 20, 8, -5. Only addition and subtraction is used to calculate the filter result.

A comparator on one threshold is used to do 2bit sign/mag format re-quantization. The threshold can be either preset value or adaptive value.

3.3 AE Buffer

AE buffer is composed by a 32bit width SRAM and companion input/output logic. The typical size of AE buffer is 64kB to 256kB depends on the demanded acquisition sensitivity. Each sample has totally 4 bits, so at 2.046MHz sample rate, 1 millisecond sample occupies 1kB of RAM.

Because each sample has 4 bits and the data width of SRAM is 32bit, so 8 samples can do read/write operation parallel on input/output interface. The read/write port is designed to work at the same time without confliction using single port SRAM. The AE buffer has the support to read sample continuously from a desired address to achieve acquisition from different code phases.

At the epoch of first sample filled into AE buffer, an event signal is sent to TE FIFO to latch the write address. The latched value is then be used to synchronize the signal in AE buffer and TE FIFO to help the AE to TE aiding process.

3.4 PRN code generator

The same PRN code generators are adopted in AE as in TE. It is further encapsulated to configure to generate different code sequence by setting 2bit system selection and 6bit SVID instead of setting initial phase. Both memory code and Weil code need to access external ROM/RAM. To minimize the logic size, AE uses the same ROM/RAM as used by TE. A simple arbiter is used with AE has lower priority than TE. This is because TE runs continuously with the continuous data output from TE FIFO, but the state machine in AE will wait for code shift register to be filled before going into next step.

3.5 Doppler search

The match filter does correlation on signal assumed to have zero IF. Because the coherent time limits the search bandwidth, different Doppler needs to be removed before input into match filter to achieve wider search bandwidth. The step between each Doppler search is called one stride. The Doppler search starts from the estimated Doppler (or from 0 if unknown) and goes to both sides. The offset of each Doppler search round will use the sequence of 0, 1, -1, 2, -2,… to multiply the stride frequency.

The Doppler remove also uses 64 entry lookup sin/cos table controlled by a 32bit NCO. The amplitude of sin/cos value is 7 and use adder to calculate the multiply result with the 1 or 3 magnitude.

3.6 Match filter

Match filter uses adder tree to calculate the dot product of the sample sequence and the code sequence. The input sample is 6bit I and 6bit Q, and there are totally 682 samples and 341 codes each repeated once to match the length of samples.

Several techniques are adopted to reduce the logic of match filter.

Because each PRN code is repeated once, so there are actually 341 pairs of samples and code instead of 682. This reduces the input data number of the adder tree by half. When samples are shifted into the sample shift register, the adjacent two samples are added together first. Which means the input of sample shift register are s_1+s_2, s_2+s_3, s_3+s_4,… instead of s_1, s_2, s_3,… In this case, the input to the adder tree of first phase will choose the odd indexed samples, then the second phase choose the even index samples as input, then odd number again. Actually, with the samples shifted, the same half of sample shift registers is chosen.

The second technique is to avoid calculating 2’s complement negative value when multiply with PRN code value -1. Because calculating negative value on 2’s complement data need an extra add one logic which is needed on all 341 sample/code pairs, we use the code bit XOR with all bits of the samples to achieve the sample/code multiplication, and put the extra add-one operation together. For a balanced PRN code, half of the code sequence is 1 and the other half is 0. This means there are totally 512 1s in each 1023 length code sequence. So 512 can be added as last step of adder tree. Because the 1s are evenly distributed in 3 segments of code sequence with 1023 length (code sequence with longer length is the same), so in order to compensate the overall value 512 evenly on 3 segment results, 160, 160 and 192 are added to the results of each segment respectively.

The third technique is to use 1bit adder tree. The adder tree usually occupies the greatest area within the match filter. Adoption of 1bit adder tree can avoid using multiple bit adders in the first several stages of adder tree. The use of 1bit adder tree also gives possibility to use it multiple times on different bit. When we have a 6bit value $b_5b_4b_3b_2b_1b_0$ represented as 2's complement format, the value can be expressed with following equation: $-b_5 \times 2^5 + b_4 \times 2^4 + b_3 \times 2^3 + b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0$. The value of b_0 to b_5 has the value of either 0 or 1. When adding a 6bit sequence together, it is equivalent to add all b_0 of the sequence then multiply with factor of 2^0, then adding all b_1 together then multiply with factor of 2^1, until b_5 using factor of -2^5. The sum of these 6 values is equal to the sum of the sequence. This means for 6bit I and 6bit Q, we totally need twelve 1bit adder trees to get the result of match filter. We can use 12 physical 1bit adder trees, or use 6 physical 1bit adder trees two time, or 3 physical 1bit adder trees 4 times depending on the tradeoff of speed and logic area. It should be noted that the factor of MSB is -2^5 instead of 2^5, so we need to use bitwise NOT of the result of the adder tree calculating b_5, then add 2^5 or 32. Put this value with the compensate values in second technique, the overall compensate value of each segments are 192, 192 and 224.

3.7 DFT and coherent sum

Coherent sum will effectively increase the acquisition sensitivity with the tradeoff of narrower acquisition bandwidth. The acquisition bandwidth is limited to about $\pm\frac{1}{T}$ with T as coherent integration time. So the acquisition bandwidth is inversely proportional to coherent time. In order to increase the acquisition speed, DFT method is adopted. The result of each millisecond will be multiplied with 4 different rotation factors and accumulate together. So the coherent sum result of 8 frequency bins will be calculated in parallel, which wider the search bandwidth. The equation used to calculate DFT result is:

$$X(k) = \sum_{n=0}^{N-1} x(i)e^{-j2\pi kn\Delta\varphi}, k = \pm 1,3,5,7$$

Here N is the length of DFT (also known as coherent sum length). In the above equation, the two results of multiplying a pair of conjugated rotation factors can be calculate together:

$$x(i)(\cos\theta \pm i \cdot \sin\theta) = (I + i \cdot Q)(\cos\theta \pm i \cdot \sin\theta)$$

$$= (I \cdot \cos\theta \mp Q \cdot \sin\theta) + i \cdot (Q \cdot \cos\theta \pm I \cdot \sin\theta)$$

In the above equation, one complex multiplication equals 4 times of real multiplication. Use different addition or subtraction on the multiplication results will get two complex results on applying two conjugate rotation factors. The frequency offsets of DFT are $\pm\Delta f$, $\pm 3\Delta f$, $\pm 5\Delta f$ and $\pm 7\Delta f$, with interval between each frequency bins at $2\Delta f$.

The rotation factor comes from a 256 entry lookup table indexed from 8MSB of 14bit NCO. The value of real part and imaginary part of rotation factor is 10bit ranging from -511 to 511.

The DFT interval control value has the following relation with the Δf value in frequency interval equation: In DFT equation, rotation angle difference between two consecutive millisecond results is $\Delta\varphi = \Delta f \cdot T = 0.001\Delta f$, So the increment value Δi (or the DFT FCW) to the rotation factor control NCO can be calculated by:

$$\frac{2\pi i}{2^{14}} = 2\pi k n \Delta\varphi = \frac{2\pi k n \Delta f}{1000}$$

$$i = \frac{2^{14} k n \Delta f}{1000}$$

$$\Delta i = \frac{i}{kn} = \frac{2^{14}\Delta f}{1000}$$

For example, if the Doppler search stride is 500Hz, then the 8 DFT results are evenly distributed within 500Hz bandwidth with interval of frequency bins $2\Delta f = \frac{500\text{Hz}}{8} = 62.5\text{Hz}$, which means $\Delta f = 31.25\text{Hz}$. Put this value into equation $\Delta i = \frac{2^{14}\Delta f}{1000}$ will get $\Delta i = 512$.

Because the coherent sum time determines the search bandwidth, so the stride needs to be adjusted according to the coherent sum time. And when the 8 DFT results place evenly within a stride, the DFT frequency control word is also determined. The interval of DFT $2\Delta f$ is 1/8 of a stride ΔF, so we have $\Delta f = \frac{\Delta F}{16}$, and then have the DFT control word $\Delta i = \frac{2^{10}\Delta F}{1000}$.

The match filter outputs 1/3 millisecond correlation result of 682 code phases, then the next 1/3 millisecond of the same 682 results and so on. A coherent SRAM is needed to accumulate 8 rotated match filter output for 682 code phases. Because each accumulation result using exp10 format which is 24bit or 3 bytes, so the overall SRAM size of 3x682x8=16kB.

After the coherent sum completed, the results of 8 bins for 682 code phases are stored in coherent RAM. And at this time, the maximum exponent is calculated to do normalization when the coherent results sent to non-coherent sum module.

The data in coherent sum RAM is frequently accessed during data processing, so the bandwidth of the RAM should be enough to avoid read/write confliction. The data bit width of coherent sum RAM is 24x8=192bits, so the data of all 8 frequency bins can be read or write together.

The coherent sum module will access the coherent sum RAM two times on doing accumulation. One read access to read the previous value and one write access to write back the accumulated value. It should be noted that the coherent sum module will not sent the result to non-coherent sum module directly after last coherent sum. Instead, the result is written back to coherent sum RAM to calculate the maximum exponent. During the first round of next coherent sum loop, the previous coherent sum result will be read out by coherent sum module and output to non-coherent sum module.

3.8 Non-coherent sum

Navigation data and NH code modulation limits the length of coherent sum. The length is also limited by the desired search bandwidth, so non-coherent sum is needed to further increase the acquisition sensitivity. Statistically, the expectation values for signals with and without signal increase at the same rate with the increase of number of non-coherent sum times. But one the other hand, at the same false alarm possibility, the possibility of signal detection will increase with larger non-coherent sum number. The effect is similar for power accumulation and amplitude accumulation, so amplitude accumulation is adopted in non-coherent sum method to make the result has smaller range.

Same as coherent sum SRAM is needed for coherent sum module, non-coherent sum module also need a SRAM to store accumulated value. The accumulated amplitude will only occupies 8bit for each frequency bin in each code phase, so the size of non-coherent sum SRAM is 682x8=5.3kB. All data in non-coherent sum RAM share the same exponent.

In the process of non-coherent accumulation, there will be possibility the sum exceeds 255 and need 9 bits. In this case, the exponent will increase by 1 and the 9bit result will be right shift 1bit to 8bit. During each round of non-coherent sum, the accumulated value will not go beyond 510 for all 682x8 results. So when the first overflow happens, we will not right shift all previous values. Instead, the index at which point the exponent increase is saved. All the following accumulation results use the new exponent and will not overflow again. When the next time the non-coherent accumulated value read out, data before this index uses old exponent which is updated exponent minus one, data after this index uses new exponent. This mechanism avoids additional access to non-coherent sum SRAM.

Non-coherent sum module will also calculate reference noise floor. During the last round of non-coherent sum, The 8 non-coherent sum results in each code phase will be averaged, and the 682 averaged value will be added together to get a 18bit result. This reference noise floor will help to determine whether signal exists by software. It should be noted that in the case of early termination, the non-coherent sum will not go to the last round and the noise floor value will be invalid in this case.

3.9 Peak sorter

Peak sorter is used to compare all non-coherent sum results and select the largest 3 within them. The results stored in peak sorter include the signal amplitude, the corresponding code phase and Doppler.

The non-coherent result with largest amplitude within 8 bins of each code phase is first selected to send to the peak sorter. If the amplitude is greater than any of the results stored in peak sorter, one of the results stored in peak sorter will be replaced. The strategy of the replacement depend on whether the Doppler or code phase of the new result adjacent to any of the exist peak in peak sorter. For searching with stride number greater than 1, the stride index will be put together with DFT index to compose a combined Doppler value.

According to the above replacement strategy, depending on whether the amplitude of the input result greater than existing peak and whether the Doppler and code phase match, the peak stored in the peak sorter after new result input will be one of the 7 cases listed in the following table (1, 2 and 3 represent previous peak, X is the new input result):

Case number	Value position
0	123
1	X12
2	1X2
3	12X
4	X23
5	X13
6	1X3

Which of the cases is adopted to do the replacement depends on the amplitude comparison result and whether the new result matches with any of the existing result. It is listed in the following table:

	No match	Match 1	Match 2	Match 3
X<3	0	0	0	0
X>1	1	4	5	1
1>X>2	2	0	6	2
2>X>3	3	0	0	3

From the above table, 16 combinations mapped to 7 replacement cases. And for each of the peak result in peak sorter, there are only selections depending on the case number: not changed, replaced with greater result and replaced with new result (peak one has only two selections). So the case number will determine the replacement strategy for each peak.

If early terminate is enabled, the peak sorter will determine whether the ratio of amplitude of greatest peak and the third peak is greater than a certain threshold after each round of non-coherent sum. If the ratio is greater than the threshold, the acquisition for the current SV is determined to success and stops the following process, the acquisition for next SV will start immediately. This helps to speed up the acquisition of string signal even longer coherent and non-coherent time and wider search bandwidth is configured.

3.10 Search parameter configuration

Search parameters for each channel (typically each channel for one SV) are put in to AE configure buffer by CPU, then CPU will start the AE after AE buffer filled. After completing the acquisition of each channel, the search result will be also written back to configure buffer. CPU can read configure buffer to get the acquisition result of each channel.

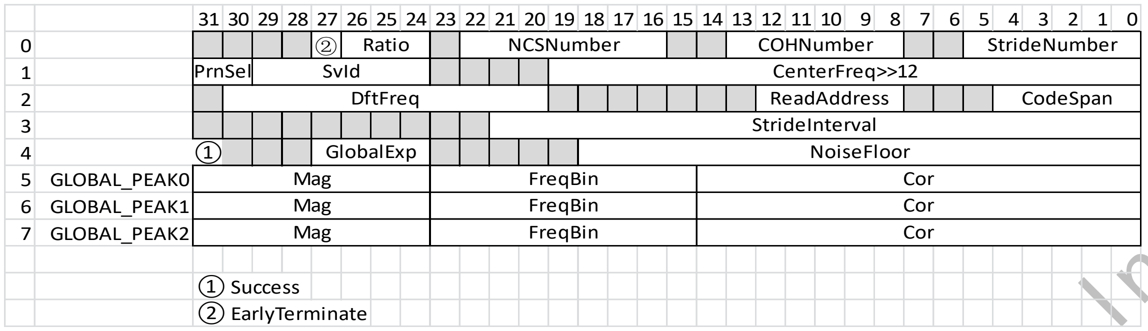
The parameters can be configured for each channel is listed in the following table:

Parameter	Range	Description	Note
PrnSel	0~3	00~11 for L1C/A, E1C, B1C and L1C respectively	
Coherent Number	1~63	Coherent sum number	
Non-coh. Number	1~127	Non-coherent sum times	
Stride Number	1~63	stride number	
Read Address	0~31	AE buffer read address offset, scale factor 682	
Code Span	0~31	Code phase search range, scale 682	
Center frequency	$-2^{19}\sim2^{19}-1$	Search center frequency	scale 0.5125/Hz
Stride Interval	$0\sim2^{22}-1$	Stride interval	scale 2099.202/Hz
SVID	1~63	SVID with 1 for SV1 until SV63	
DFT Frequency	$0\sim2^{11}-1$	DFT interval	scale 16.384/Hz
Early Terminate	0/1	Whether early terminate enabled	
Peak Ratio	0~7	Early terminate threshold, 0~7 for 1.125~2 with step at 0.125	

The above parameters occupies 4 DWORD in configuration buffer, combined with 4 DWORD acquisition result, each channel occupies 8 DOWRD. If AE is designed to have maximum 32 acquisition channels, the overall size of AE configure buffer is 32x8x32bit=1kB.

If more than 32 acquisition channels are needed, for example the number of satellite to be searched greater than 32 or need multiple channels for wider Doppler search range, the CPU can set new parameters in configure buffer and start new task.

The placement of parameters and search result for each channel is shown in the following figure:



		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						②	Ratio				NCSNumber									COHNumber								StrideNumber					
1		PrnSel		SvId										CenterFreq>>12																		
2			DftFreq																		ReadAddress								CodeSpan				
3												StrideInterval																					
4		①				GlobalExp									NoiseFloor																		
5	GLOBAL_PEAK0	Mag								FreqBin									Cor														
6	GLOBAL_PEAK1	Mag								FreqBin									Cor														
7	GLOBAL_PEAK2	Mag								FreqBin									Cor														
		① Success																															
		② EarlyTerminate																															

Before starting the AE, the AE buffer needs to be filled first. Whether the filling of AE buffer has completed is indicated in the status register. After the AE completes the acquisition, an AE interrupt is generated; CPU can read out the acquisition result and performs following operation depending on the result.

For different signals, the length of PRN code differs. So number of code phases need to be search is also different. The code span field is the number of code phases need to be searched with scale factor of 682. So if all code phases are to be searched, its value need to be set to 3 for GPS L1C/A, 12 for Galileo E1 and 30 for B1C and L1C. The read address field determines the signal offset in AE buffer start to search. For full code phase search, read address can be set to 0. If prior information is available to limit the code search range, read address and code span can be set to different value to determine start and range of code phases to search (e.g. during signal reacquisition).

3.11 Search result resolving

The three greatest peak occupies the last 3 DWORD in configure buffer of each channel. The magnitude occupies 8 bits (the exponent put in DWORD 4), the Doppler occupies 9 bits and the code phase occupies 15 bits. The meaning of the code phase is the address offset of the sample that aligned to the first bit of the PRN sequence. And the 9bit Doppler composed by 6bit stride offset and 3bit DFT index. The DFT interval is usually set to 1/8 of a stride interval. The 8 DFT bins are evenly placed on the center of a stride, so DFT bin index 0~7 means -3.5~3.5 times of DFT interval from the center of each stride. So the Doppler search result need to first minus 3.5, and then multiply with DFT interval to calculate the Doppler frequency in Hz. For example, when stride interval set to 500Hz (and corresponding DFT interval set to 500Hz/8=62.5Hz), -7 in Doppler field means (-7 – 3.5)x62.5Hz = -656.25Hz, 23 in Doppler field means (23 – 3.5)x62.5Hz = 1218.75Hz.

4. Accessory modules

Accessory modules are modules in GNSS receiver that cooperate with signal processing module (AE and TE) to implement certain functions. One typical accessory module is PPS.

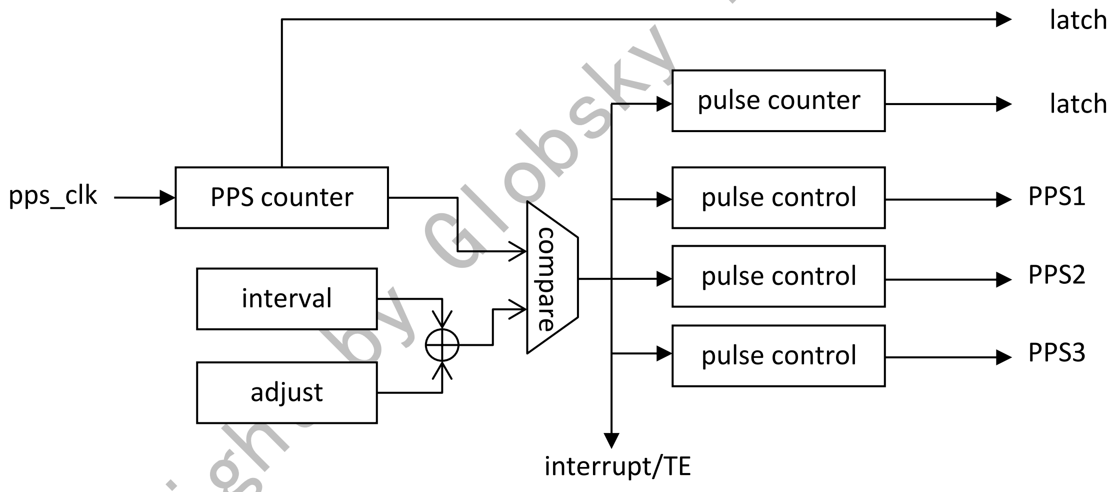
4.1 PPS module

The function of PPS module is to output pulse signal that align the pulse edge to the boundary of UTC second or other time mark. The PPS module driven by higher frequency clock has better resolution on the pulse edge, so the PPS module is designed to operate by clock asynchronous to system clock or the clock of GNSS baseband.

Considering the need to provide pulses align to different time system, the PPS is designed to be able to output three pulses with different delay, polarity of edge and the width of pulse. These three pulses are controlled by one master counter, and can be used to align to different time systems (e.g. UTC, GPS time and BDS time).

4.1.1 PPS function design

The block diagram of PPS module is shown below:



The PPS counter is driven by PPS clock. When the counter reaches the PPS pulse period, a PPS event is generated. The PPS event will trigger following operations: increase the PPS pulse number, generate PPS interrupt, latch the write address of TE FIFO and trigger the pulse control logic. There are three sets of pulse control logic, each has different delay to output pulse, different pulse width and pulse polarity control.

There is an interval adjust register that adjusts the edge of all pulses. The adjust register is written by CPU and effect only once.

At the event of CPU command or external event happens, the PPS counter and pulse counter will be latched. The maximum clock frequency is designed to be less than 1GHz, so the width of the counter is set to 31bit to compare with the sum of 30bit interval and 30bit adjustment. Pulse counter is 8bit to ensure the value to be unique in 256 seconds.

4.1.2 Cross clock domain

The PPS module operates in PPS clock domain, but some signals interacting with PPS are controlled by CPU or GNSS baseband processor and are in system clock domain. So the access to the control registers and state registers need cross clock domain design.

The read operation to control register and state register can be accessed directly by CPU even they are in different clock domain. This is because the control register is changed only by CPU, and the state registers accessible by CPU are all latched values. Software will make sure there will be enough hold time for these registers and the contents won't change during read. So the values of the registers are considered to be static signal.

When CPU write control registers, the address and the value of the registers are first latched in system clock domain together with write valid state toggled. These signals will pass a two stage latch after go into PPS clock domain. The latch ensures the stability and consistency of the signal. The logic in PPS domain will detect the toggle of the latched value of write valid state and determine whether a new write is effect. As long as the PPS clock is faster than the system clock, this mechanism gives stable and reliable cross clock domain access.

4.1.3 PPS control procedure

The following steps are performed by CPU to achieve initial alignment of the PPS pulse:

Step 1: After the signal is tracked and PVT gets the position fix, PPS module is enabled with PPS_CTRL register.

Step 2: When PPS event generated, TE_FIFO_LWADDR_PPS register in TE FIFO holds the latched TE FIFO write address at the epoch of the event. PPS interrupt will help to inform CPU when this event happens.

Step 3: By using clock error calculated in PVT, aligned TE FIFO read/write address, system time at the epoch of PPS event can be calculated. Convert the desired time difference to be adjusted to number of PPS clock, and write the corresponding value to PPS_PULSE_ADJUST register.

Step 4: Calculate PPS_PUSLE_INTERVAL using clock drifting if PPS clock and system clock have the same source.

Step 5: Optionally repeat step 2 and 3 to refine the pulse edge.

Step 6: Set the pulse width and polarity register and enable the pulse output.

Following points are to be noted on control the PPS: Firstly, the pulse control module can only has positive delay, so the PPS event should be set earlier than any pulse edge (it is better to be hundreds of ns or 1us earlier). Secondly, change to any register value is better to be performed as soon as possible after PPS event happens. Thirdly, adjustment to PPS_PULSE_ADJUST will affect all pulses. Use delay settings in each pulse control module to adjust the pulse edge individually.

The PPS is also capable to latch the PPS counter and pulse counter by CPU command and external event. This is a redundant function design to provide another way of time synchronization through different modules.