**Key Features:**

Time resolution: 50 ps

Pulse width: 7.5 ns-2.6 s

Input low voltage: -5.2 V to +0.4 V

Input high voltage: +0.6 V to 3.5 V

Output low voltage: 0 V

Output high voltage: 3.3 V

Working voltage: 12 V

Maximum power consumption: 12 W

Hysteresis: 25 mV to 75 mV, Typ: 50 mV

Max number of channels: 8

Sequence storage memory: 4 GB

Coupling: DC 50 Ohm

Short-term jitter: < 15 ps

Long-term jitter: < 25 ps (Pulse width = 500 ms)

High dynamic range and continuous output

High stability and low long-term jitter

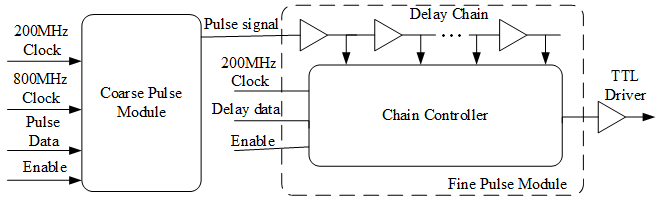
High precise synchronization

**General Description:**

ASG-GT50-C serials are high performance pulse/delay generators named Arbitrary Sequence Generators (ASG), equipped with high speed read-out counters. The device supports continuous pulse output with a 7.5 ns minimum pulse width and with non-dead-time. The time resolution of each rising and falling edge of the pulse signal is 50 ps. A single digital pulse data consist of two 32-bit data for coarse time duration of both logic ‘1’ and ‘0’, and two 8-bit data for fine time delay of the rising and falling edge. A DDR3 memory with 1 GB (4 GB available) Capacity is integrated for pulse data storage. The large memory capacity allows a maximum pulse number exceeding 10^7 for each single channel.

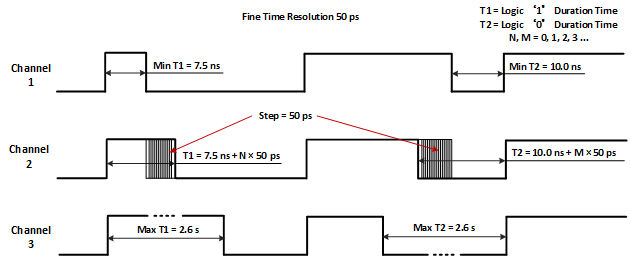
**Appearance and Dimensions:**

**Introduction:**



The pulse generation module is implemented using a time interpolating method, the functional block diagram of a single pulse channel is shown in figure above. The coarse pulse module operates with a 200 MHz clock, and outputs pulse signals with an 800 MHz clock, which contributes to a 1.25 ns time resolution. A second stage fine pulse module is implemented to improve the time resolution of the fast pulse generation. The pulse signal will go through the delay chain before being output, and the output position can be adjusted in real time when operating. The averaged cell delay of the delay chain is 50 ps, thus the fine time resolution of the pulse signal is 50 ps.

The figure below shows an example timing diagram of the pulse signals. The pulse width of signals from the coarse pulse module is integer times of the 800 MHz clock period. After going through the delay chain, the rising and falling edges of the pulse signals will be given a respective fine delay relative to the previous clock rising edge.



**Attention:**

* Connect at least one output port before powering up.
* The power adapter for the device must be connected to 220 VAC. Or use 12 VDC power supply with the power supply cord.

**MAC OSX:**

1、Startup

For convenience, all download steps below can be skipped by asking technique support for the files.

Download the SDK from <http://www.cypress.com/documentation/software-and-drivers/ez-usb-fx3-software-development-kit>

Uncompress the SDK file：FX3\_SDK\_MacOS\_v1.3.3.tar.gz

And uncompress the file：cyusb\_mac\_1.0.tar.gz

Open the 'docs' folder，and refer to 'cyusb\_mac\_user\_guide.pdf'. You can start from the section 2.1. Now open ‘Terminal’.

1. Install the library 'libusb'.
2. Download the library ‘libusb’from the link: <http://sourceforge.net/projects/libusb/files/libusb-1.0/>.
3. Extract the compressed file, then we get a folder (such as libusb-1.0.21)
4. Enter the folder ‘libusb-1.0.21’, run ‘./configure’ in the Terminal.
5. Then run ‘make’ to compile the package.
6. Type ‘make install’ to install the programs and any data files and documentation. When installing into a prefix owned by root, it is recommended that the package be configured and built as a regular user, and only the ‘make install’ phase executed with root privileges. That mean you may run the ‘sudo make install’ instead of ‘make install’.
7. If no error, continue to ‘B’ step.
8. Enter the 'cyusb\_mac\_1.0/lib' folder and run 'make' command in the terminal. Then copy the '\*.dylib' file to the /usr/local/lib.
9. If ‘make’ fails with something like ‘can not find libusb.h’ or ‘can not find –lusb-1.0’, now go to step (2)
10. In this folder, there is a ‘Makefile’ file. Open it. Modified the second line to “g++ -dynamiclib libcyusb.c **–I /usr/local/include/** -o libcyusb.0.1.dylib **–L/usr/local/lib/** -l usb-1.0”. And then run ‘make’ again.
11. Run ‘cp \*.dylib /usr/local/lib’ command to copy the files.
12. Copy the ‘cyusb\_mac\_1.0/configs/cyusb.conf’ to ‘/etc’ folder.
    1. If you just finish the step ‘B’. Run ‘cd ..’ and ‘cd configs’ to go to the ‘configs’ folder.
    2. Then run ‘sudo cp cyusb.conf /etc’ to copy the file. And this command need root privileges, so we need to use ‘sudo’.
13. Go to the ‘cyusb\_mac\_1.0/examples’ folder and use the ‘cybulk\_writer.c’ we provided to replace the same file in the ‘cyusb\_mac\_1.0/examples’ folder.
14. Run ‘make’ to build all executable files. And if in the step ‘B’ you meet the errors, it may occur again. Please modified the ‘Makefile’ again. Each line start with ‘g++’ must be modified like step B.(2).
    1. For example, the second line will be
    2. ‘g++ -o cyisowrite\_test **–I /usr/local/include/** **–L/usr/local/lib/** cyisowrite\_test.c -l cyusb -l usb-1.0’.
    3. Please know the ‘-I’ option specify the location of the header files and the ‘-L’ option specify the location of the library files.
15. Copy the ‘write.dat’ file we provided to the ‘cyusb\_mac\_1.0/examples’ folder.
16. Run './cybulk\_writer 512' command in the terminal. If no errors, the install is succeed.
17. If you install the Python interpreter and package 'numpy', please copy the ‘macosx\_pulse\_api.py’ file we provided to the ‘cyusb\_mac\_1.0/examples’ folder. And run ‘python macosx\_pulse\_api.py’ to see if Python API works.
    1. Please know the version of the Python interpreter must be 2.7.X, not 3.X.X.
    2. No error message means it is working well.
18. Now you need to know how to use the Python API. Before that, you may want to learn the grammer of the Python.
19. If you want to use another language to run the ASG, please use command line to call the Python script.

2、Python API Introduction

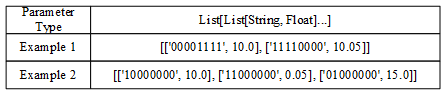
We also provide API written in Python. The filename is 'macosx\_pulse\_api.py'. Before you running the script, you must install Python interpreter and package 'numpy'. The Python script use the program 'cybulk\_writer' to download data to the device. So they must exist in the same folder.

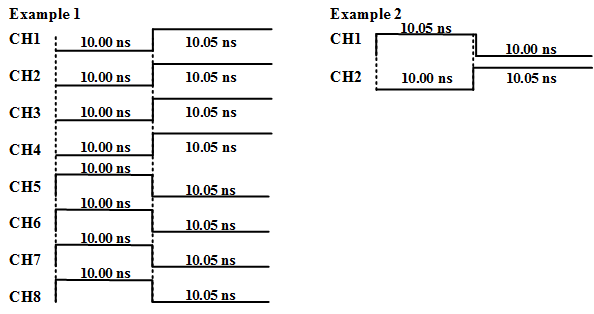
For the example to use the Python script, please refer to the functions 'delay\_chain\_test' and 'random\_pulse\_test' in the script. You can also run the script in the terminal. If all is configured well, the device will output signal.

We provide 3 API functions for the user: start, stop and PB\_type\_program. For more information, please refer to the comment in the functionds.

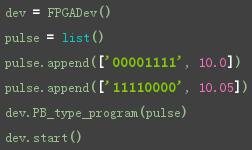
|  |  |
| --- | --- |
| Function Name | Explanation |
| start | Start pulse playing. |
| stop | Stop the playing. |
| PB\_type\_program | This function generate the processed bytes. And write to the ‘write.dat’. |

Parameter for the function 'PB\_type\_program':





API calling example:



If you are using other programming languages, you can call the python script through the commandline with:

1. **python macosx\_pulse\_api.py**

This command will download the pulse sequence data describe in the ‘pulse.txt’ to the device and start playing.

1. **python macosx\_pulse\_api.py –-stop**

This command will stop playing.

3、Understand the Data Describing Pulse Sequence

Assume that is a command: **BBBBBBBB, I, I, F**.

The first 8 characters ‘**BBBBBBBB**’ describe the states of the 8 output channels, and ‘0’ represents the low electric level and ‘1’ is high electric level. And each character represents an output channel. It indicates OUT1 to OUT8 channel from left character to right character.

The middle two integer ‘**I**’ is for compatible with the PulseBlaster and unused now, just fill with 0.

The last floating number ‘**F**’ represents the duration of this command.

And pulse sequence is consist of many commands. All commands will be executed on by one.

Let we look at an example.

Command 1: 11110000, 0, 0, 10.0

Command 2: 11111000, 0, 0, 10.0

Command 3: 11111000, 0, 0, 0.05

In this example, the OUT5 will remain low level for 10 ns and then turn to high level for 10.05 ns.

After calling the Python script, the data will be converted to the processed bytes, which I will discuss in the next.

We use 10 bytes (80-bit) to describe a pulse with high level and low level. Each 80-bit data includes 32-bit high level time data, 32-bit low level time data, 5-bit leading edge delay data, 5-bit trailing edge delay data, and 6 reserved bits.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 32-bit | 32-bit |  | 5-bit |  | 5-bit |
| High  Level | Low  Level |  | Rising  Edge |  | Falling  Edge |

The least bit of the 32-bit data represents 0.625 ns for high level time or low level time. And the 5-bit data adjust the edge of the pulse with 50 ps resolution.

**Be careful, the adjustment of the falling edge will decrease the later negative pulse width by the same value. And this is the same with the rising edge. Please see the diagram below:**



For each address in the DDR3 memory, it can store 512-bit data. See below:

|  |  |  |  |
| --- | --- | --- | --- |
| 32-bit | 80-bit | … | 80-bit |
| Record the valid pulse in this address |  |  |  |

And the format of the 32-bit data is below:

|  |  |  |  |
| --- | --- | --- | --- |
| 8-bit | 8-bit | 8-bit | 8-bit |
| 0XXX0000 | 00000000 | 00000000 | 00000000 |
| XXX from 000 to 110 |  |  |  |

And we use different blocks of address to store the data of different channels

**Basics of the Communication:**

The ASG-GT50-C based on the chip of the Cypress Company. And the company provide the Software Development Kit (SDK) for us to communicate with the device. However, in Linux and MAC, the SDK must be based on the ‘libusb’ library.



Explanation:

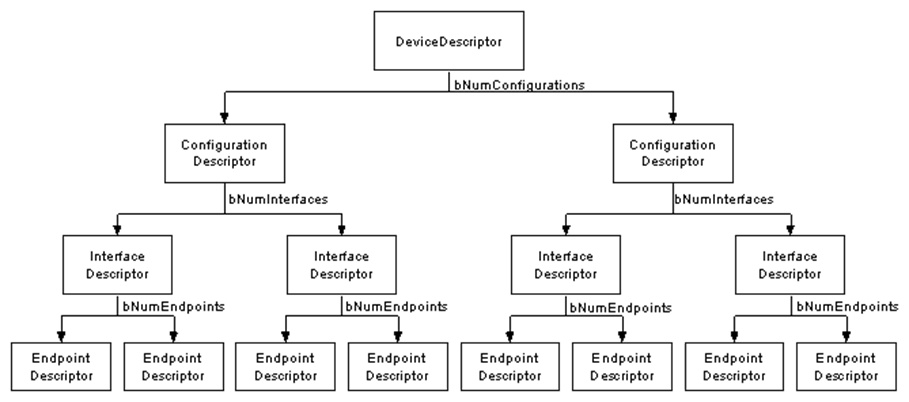
**Libusb library**: libusb is a C library that provides generic access to USB devices. It is intended to be used by developers to facilitate the production of applications that communicate with USB hardware. [[More information](http://libusb.info/)]

**SDK**: the SDK provide higher level function for user to use. It means you can integrate the code into your project. And the file ‘cybulk\_writer’ will open the device and find the first ‘out’ endpoint, and then send the data in the file ‘write.dat’ to the device, and then close the device.

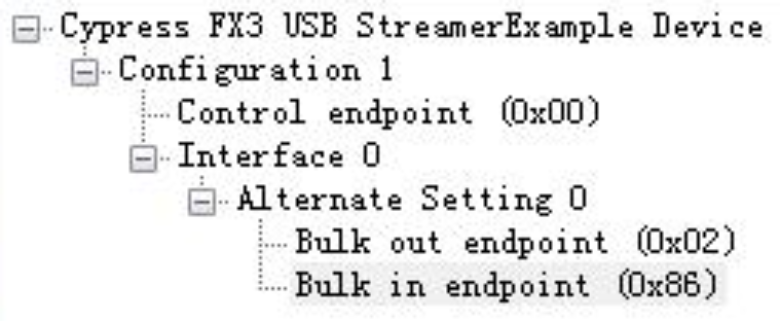
**Python script**: the script converts the data to processed bytes. The key function is ‘PB\_type\_program’ and ‘PB\_input\_create\_buf’ if you want to convert to another programming language.

For a USB device, it may contain several Configuration, Interface and Endpoint. And the explanation is below:

* Configuration describes function collections of the device.
* Interface describes the type of a function.
* Endpoint is virtual transfer port and it has fixed data transfer direction, except the Endpoint 0.



And our device has just one Configuration and one Inteface. The Control endpoint is not intended for users, please not use it. The Bulk endpoints are used for transferring data to and from the device. And ‘out’ endpoint indicates the data transfer from computer to the device, while ‘in’ endpoint is on the contrary.



For the application of playing pulse sequence, users only need the ‘out’ endpoint.

And the most important thing is, the number of transferring bytes must be EVEN! The feature is due to the design of the hardware.

For the API of the SDK, you can refer to ‘cyusb\_mac\_1.0\docs\cyusb\_mac\_programmers\_guide.pdf’ for more information.