ON Semiconductor

Is Now

To learn more about onsemi™, please visit our website at
www.onsemi.com

AND9351/D

PYTHON Frame Rate Calculator (PFC) User's Manual

ON Semiconductor®

www.onsemi.com

APPLICATION NOTE

Purpose

The purpose of the PYTHON Frame Rate Calculator (PFC) is to provide a tool which Field Applications Engineers can use to help customers set up their PYTHON sensors to achieve a desired frame rate and be able to come up with values to program into sensor registers. Also, it can be used to see if the desired frame rate is achievable.

Program Distribution

The program is distributed as a single exe with no installer. It is available in 32 or 64 bit versions.

Sensor Registers the Program Uses

The following is a list of PYTHON registers which PFC settings effect. If you import/export a Python scripts PFC will extract or populate information from these registers:

Table 1. SENSOR REGISTERS

OFFSET	PURPOSE	COMMENT
0	Chip ID	Chip family (P1300, P500, XK)
1	Chip Info	Identifies sensor within family (IE P500)
32	Clock Generator Config: Bits 5:4 – mux_mode	Sets down mux: 0 = all, 1 = /2, 2 = /4, 3 = /8
192	Sequencer Config: Bits 2:2 – ROT Mode Bits 7:7 – Subsample Enb Bits 8:8 – Bin Enb	 0 = NROT, 1 = ZROT 0 = off, 1 = on 0 = off, 1 = on
193	XSM Delay Bits 15:8	Per line delay added
194	Integration Control: Bits 11:10 – Subsample mode Bits 13:12 – Bin Mode	 0=XY, 1=X, 2=Y, 3=XY 0=XY, 1=X, 2=Y, 3=XY
195	Active ROIs	Each bit is enable for ROI. B15:0 enable ROI15:0
196	Active ROIs – XK upper	B31:16 used to enable ROI31:16
197	Black Lines Generated Bits 7:0	Tells sensor how many electronic black lines to generate
199	Mult_Timer	Resolution for timing frame time and exposure. Expressed in master clock periods. We set this to 72 for a resolution of 1 μs.
200	fr_length	Frame length, this sets frame rate.
216	FOT base	Offset added to 384 for FOT base.
220	ROT base	Offset added to 384 for ROT base.
256	ROI0 X: Bits 7:0 – start Bits 15:8 – end	Start and end expressed in kernels not pixels
257	ROI0 Y Start	
258	ROI0 Y End	
259 – 351	Same as above for ROIs 2 – 31	1300 family has 8 ROIs, 5000 family 16 and XK 32 So number of registers valid in this range depend on sensor type.

Using Program

This section talks about how to use the program to create regions of interest (ROIs) which allow you to achieve the image sizes and frame rates you desire. To make things easier, this section deals with a single ROI at a time and no file IO.

In General

Editing Parameters

In PFC any parameter which has a white box is non−changeable. Any parameter you can change will have a colored box. With colored boxes, if you enter an invalid number, the edit box will turn red and the main information box will contain the error condition:

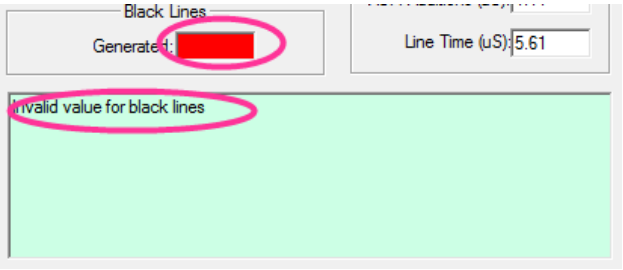

Figure 1.

Parameter Storage

Most parameters are non−volatile and they are stored in the registry. They can also be stored in files but file IO will be discussed later.

Program Version

The program version can be retrieved by doing Help → About:

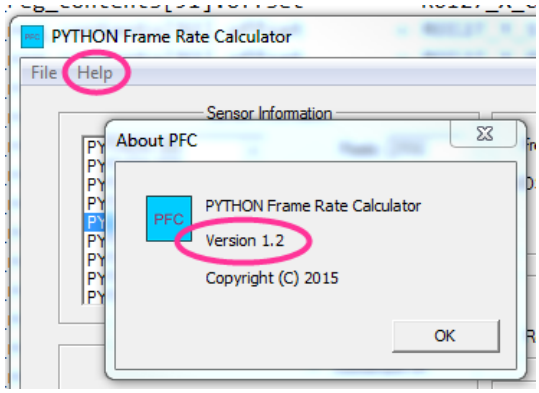

Figure 2.

Information Box

The information box is the main communications method to the user. Normally the information box will only have 1 line in it but sometimes it takes several lines to explain error conditions. In this case the box has vertical scrolling and you can scroll up/down to look at information.

Figure 3.

ROI Display

The ROI display is similar to the actual sensor output. You can configure any of the 32 possible ROIs but you will not see it unless you enable it. So, if you want to see the results of a single ROI you are working on in terms of array area it resides in or its' individual effect on frame rate just enable that ROI:

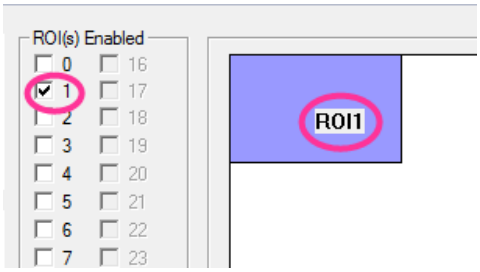

Figure 4.

ROI Configure

To **configure** an ROI select its radio button in the **configuration section**. Configuring an ROI will not affect sensor frame rate unless you enable it. If you want to configure the sensor from several image sizes and frame rates then click on the ROI in the configuration area and then just enable that one ROI in the enable section:

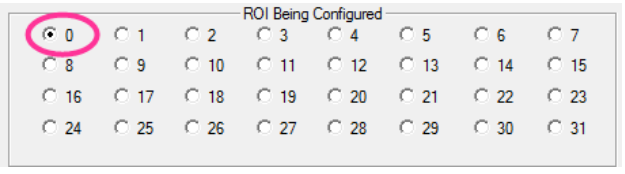

Figure 5.

Sensor Selection

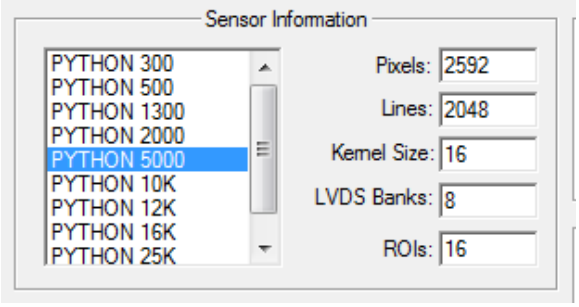

Figure 6.

This section of the GUI allows you to select your sensor of interest. This section may end up helping you determine what PYTHON sensor you need to achieve any speed requirements etc. you may have.

Select a sensor from the list box. The properties associated with the sensor are retrieved from a local database. None of these properties are changeable and therefore they are white in color.

Your latest sensor selection is saved in an internal database and also in the WINDOWS registry. The next time you open PFC your sensor should show up.

Please note: changing sensors may change your ROI definitions if the kernel size and sensor area changes. You may have to redefine some ROIs.

Master Clock

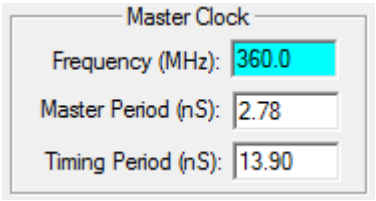

Figure 7.

The master clock, as the name implies, is the main clock in frequency. The maximum for PYTHON 300/500/1300/2000/5000 is 72 MHz and for PYTHON XK it is 360 MHz. For XK the 360 MHz is not used for timing XSM delay and fr_length etc. There is an internal divide by 5 for that. Because of this, a timing period display was added to avoid confusion. So you will see the master clock period and a timing period. For 300 through 5000 the two will be the same. For XK the timing period will be 5X larger than master clock period. The master frequency is changeable and that is why it is cyan. The master clock period for the frequency is also displayed and this period effects the time of everything in the calculator.

LVDS Banks Used

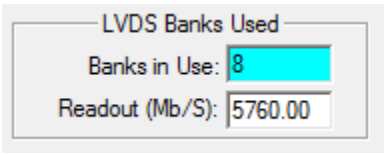

Figure 8.

You can change the number of LVDS banks the sensor uses. The sensor box tells you the maximum number but you can use less if you want to simplify your FPGA capture subsystem. Using less will decrease your readout rate and decrease frame rate. All sensors support 2x, 4x and 8x down muxing except the P1300 family. There you are limited to 4,2,1 LVDS banks.

PFC knows what the LVDS capabilities are for each sensor and it will turn the **LVDS used** edit box red if you choose a bad value.

Mult Timer

In PYTHON the mult timer register allows you to establish a resolution for timing things like exposure or frame length. By default we set up a convenient resolution of 1 μs by setting mult_timer = 72. This is 72 14 ns timing periods. By using a 1 μs resolution it is easier to look at the exposure and frame length registers and figure out what they are set to. For example, a value of 12,000 is 12 ms.

Changing this setting effects fr_length in the calculator which in turn effects frame rate.

Resultant Image

The resultant image is the image size the sensor will output after configuring your ROIs and subsampling. This section is displayed by PFC after it is calculated from the ROI and subsample parameters. The one field which can be edited is bit depth. Bit depth does not have an effect on frame rate because the LVDS clock for 8 bit is reduced to the point where the lower bit depth and LVDS clock rate provide the same frame rate as 10 bit mode with the full LVDS clock rate.

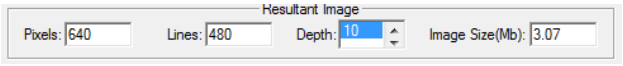

Figure 9.

Image Subsample Options

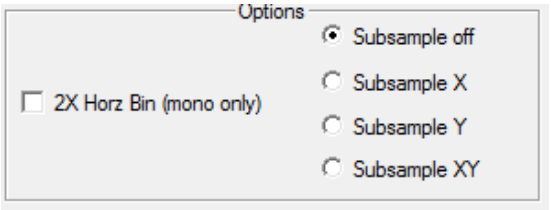

Figure 10.

Here there are two options: bin and subsample. The bin is actually an averaging in the analog domain so there is not

increase in output but there is noise reduction and frame rate improvement. The specification indicates binning is possible in the Y direction if the sensor is not running pipelined. Most of the time sensors will be run pipelined so the binY option is not allowed in PFC. BinX will cut the image width by 2 and improve frame rate.

Subsampling is actually decimation. You can subsample in both directions. You will get a frame rate increase in both directions.

Row Overhead Time Mode

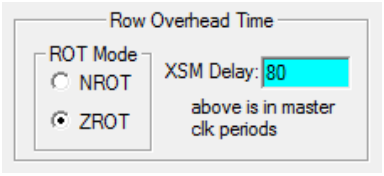

Figure 11.

There are 3 ROT modes: Normal ROT, Zero ROT and non−zero ROT.

Normal ROT, NROT has ROT sequential with readout:

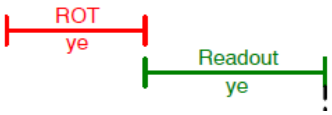

Figure 12.

Zero ROT places ROT in parallel with readout so effectively ROT does not affect line time:

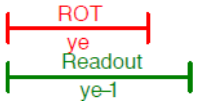

Figure 13.

Non−zero ROT, is ZROT with line delay or XSM delay. The XSM delay does affect line time and expands horizontal blanking. NZROT mode can be useful if you need HBlank time for your capture subsystem to finish capturing a line.

ROT

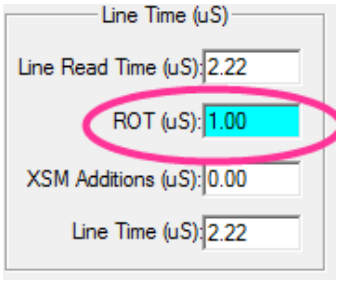

Figure 14.

By default, the row overhead time native to a specific sensor is stored internally and the ROT box will be updated when you change sensors. You are also allowed to change the ROT value. There is a chance PFC may not have the correct ROT for a special sequencer program you are running and you may want to enter the value manually.

The ROT can also be updated by importing an INI file or Python script file.

FOT

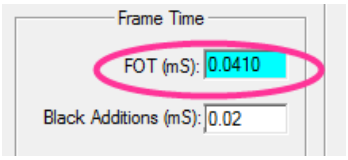

Figure 15.

By default, the frame overhead time native to a specific sensor is stored internally and the FOT box will be updated when you change sensors. You are also allowed to change the FOT value. There is a chance PFC may not have the correct FOT for a special sequencer program you are running and you may want to enter the value manually.

The FOT can also be updated by importing an INI file or Python script file.

Black Lines Generated

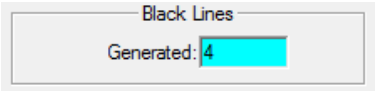

Figure 16.

PYTHON sensor are capable of generating electronic black lines. These lines are used for correction. In the register set you can generate from 1 to 255. You can also choose to ignore the first N lines. Each black line generated is the full width of the sensor regardless of ROI size. It is also subject to ROT mode and XSM delay. Time−wise, each black line is equivalent to a full resolution width line and will decrease frame rate. PFC automatically loads the black line setup for each sensor. This can be edited by hand or by doing file IO.

ROI Programming

You can configure up to 32 ROIs. PFC knows how many ROIs a sensor supports and will disable the configuration and enable buttons for the invalid ROIs. It will also disable and uncheck ROI which are invalid.

To configure and ROI select the ROI in the **configuration** section:

Figure 17.

To be able to see the operations you are applying to just this ROI disable all other ROIs and enable just the one you are configuring:

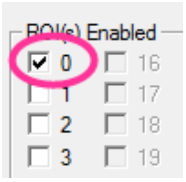

Figure 18.

If you want to set up an ROI for a known image size you can do this using the **common sizes** box. This will create an ROI of the selected size which is centered in the field of view (FOV).

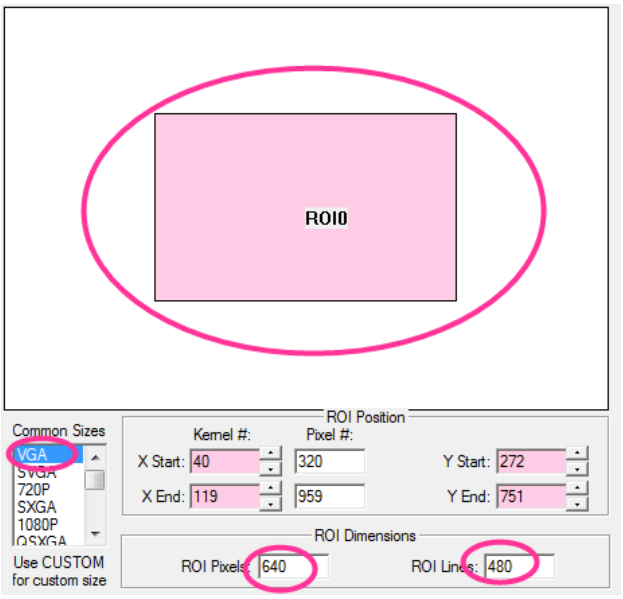

Figure 19.

If you then want to change the position you may. There are 6 parameters to set for an ROI:

- Xstart
- Xend
- Ystart
- Yend
- Pixels
- Lines

The X parameters affect the position of the ROI in the image array horizontally and they are expressed in kernels. Likewise the Y parameters affect placement vertically. The sensor information box tells you how many pixels there are in a kernel. This values changes based on what PYTHON family you are working with.

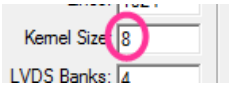

Figure 20.

If you are using a common size and change the XY parameters then PFC will calculate the other parameters for you to maintain the common width or height. You can manually type an entry into the parameter box or use the buddy control to increment or decrement values. In either case PFC will make sure you do not go outside the boundaries of the array.

In Common Size mode the pixels and lines boxes are not editable. If you want full control you can select custom size and then the pixels and lines boxes can be edited. In this mode size is not maintained by PFC and you are free to make any size you want:

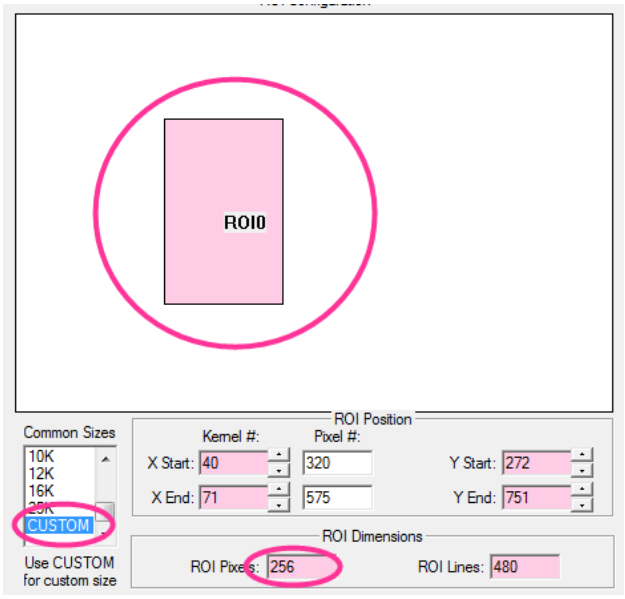

Figure 21.

Line Time Section

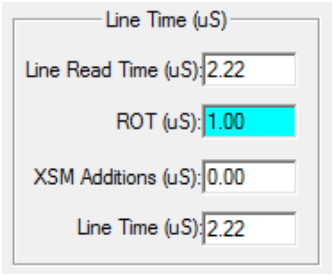

Figure 22.

The line time section groups parameters associated with line time. The image width and readout rate will determine line readout time. In addition, ROT and XSM could increase the line time if enabled. In this section the parameters are for the resultant image size using the ROI you have programmed and it may not be the line time for black lines which are always full imager width.

The ROT value is extracted from an internal database for the sensor but may be edited by the user. If you do not save this in an INI file it will be lost the next time you open PFC.

Frame Time Section

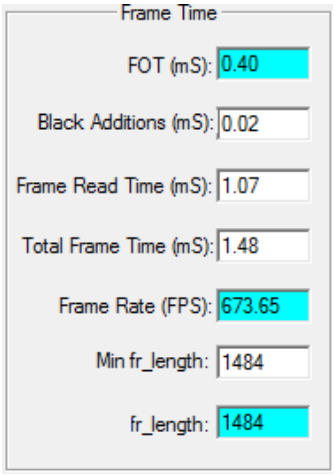

Figure 23.

This section groups information and parameters associated with frame time and rate. Most of the parameters are calculated and populated by PFC and cannot be edited in this section (but can be in other sections). The user is free to edit FOT. This value is retrieved from an internal database when the sensor model is selected. However, any setup changes you have can be saved to a Python script or ini file. That will be discussed in the File IO section.

Even though PFC calculates the frame time and frame rate you are free to override the frame rate by entering values in the fr_length or frame rate boxes. The frame rate value must be less than the calculated value and the fr_legth must be greater than the minimum fr_length. By editing fr_length or frame rate you can reduce the frame rate down to a value you wish. Reducing frame rate also gives you more potential exposure time.

Python Script Files

Under the file menu you can import or export a Python script. The export will take the current PFC setup and create a Python script which writes to the appropriate sensor registers.

Importing reads in a Python script. The script could be from a previous PFC export or it could be a register dump from SensorStudio or just a script you wrote. When reading the script in PFC will using the register contents defined in the script to populate settings.

If you are using SensorStudio you can save a PFC setup as a Python script and then run it in SensorStudio. There is one thing to be aware of – the PYTHON XK sensors need some XSM delay in order to complete FPGA line capture. You need to preserve that delay:

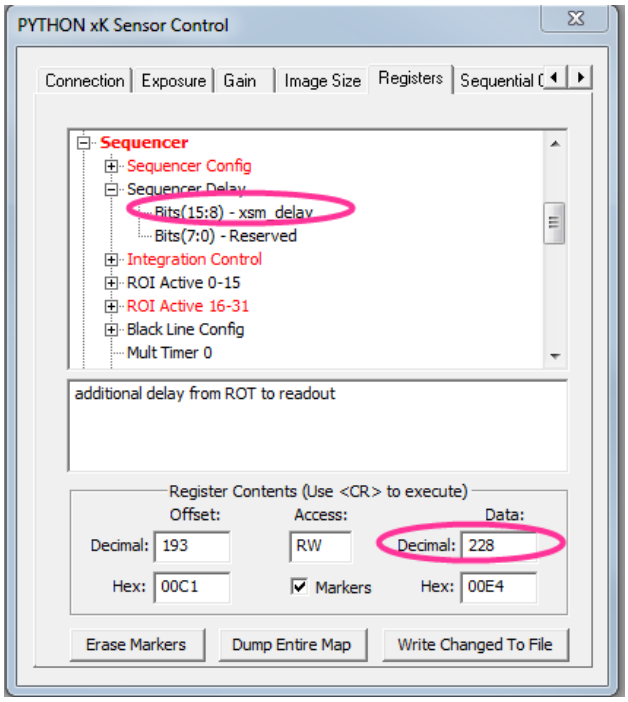

Figure 24.

The 300/500/1300/2000 and 5000 do not need delay to capture but the startup script for 2000/5000 places the sensor in NZROT mode using an XSM delay of 80. This value could change in the future if the startup register setup changes so you should always use SensorStudio to examine XSM delay if you plan on exporting your settings to a Python script.

The exported script uses SensorWrite() for full register writes and WriteSPI_B() for bitfields. So, you do not have to worry about this script altering your other register bitfields.

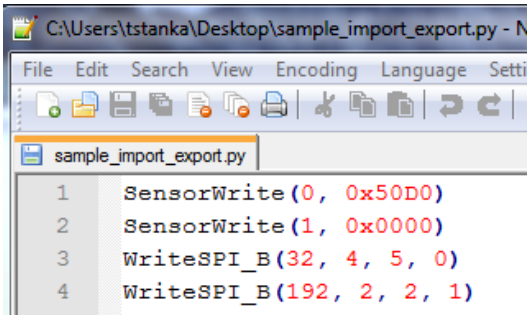

Figure 25.

You can export or import Python scripts. This means you can use a Python script to save your setups by simply exporting and then importing.

One important note. You do not need to use a Python script which writes to all the registers. PFC just looks for the registers in the file and if they are not there it does not touch setting associated with that register. For example, if the XSM delay register is not in the script PFC will simply leave the XSM delay setting alone.

INI Files

If you are not using SensorStudio you might not want to bother with Python scripts. For these situations PFC supports initialization files. These files are just text files and they are easier to read and understand and need many less lines to save a setup. Because they are easy to understand users may decide to export a setup into an INI file and then make copies of it to alter some parameters so they can import in new setups.

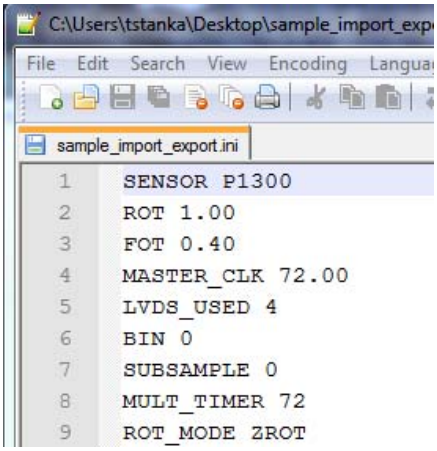

Figure 26.

Like a script file, the INI file does not have to have all the parameters defined. For example, if you just want to alter ROT you could create an INI file with just the line:

ROT 1.20

PUBLICATION ORDERING INFORMATION

LITERATURE FULFILLMENT:
Literature Distribution Center for ON Semiconductor
19521 E. 32nd Pkwy, Aurora, Colorado 80011 USA
Phone: 303−675−2175 or 800−344−3860 Toll Free USA/Canada
Fax: 303−675−2176 or 800−344−3867 Toll Free USA/Canada
Email: orderlit@onsemi.com

N. American Technical Support: 800−282−9855 Toll Free USA/Canada
Europe, Middle East and Africa Technical Support:
Phone: 421 33 790 2910
Japan Customer Focus Center
Phone: 81−3−5817−1050

ON Semiconductor Website: **www.onsemi.com**

Order Literature: http://www.onsemi.com/orderlit

For additional information, please contact your local Sales Representative