Driving the STAR and HAS2 Rolling Shutter Sensors

AND9464/D

Scope

This application note is intended as an introduction to engineers that for the first time work with rolling shutter sensors with an external sequencer – such as the STAR1000, STAR250 or HAS2 devices from **onsemi**.

Important: Before Any Code is Written

Code Planning

A very significant share of the STAR or HAS−related questions that arrive at **onsemi** support desks are from users that started working with the sample code provided by **onsemi**. In a typical case, the user ran into problems while adapting the code to their application.

This likely happens due to that the sample code tempts the user to skip the planning and abstraction phase of code writing by providing a ready−made and semi−working code set.

A more productive working scheme is not unlikely to be to use the sample code only as reference while following normal development procedures with an analysis, abstraction and planning phase.

Reducing Complexity by Increasing the Level of Abstraction

In order to be easily debugged, the code written to run a sequencer should have some level of abstraction that can be shown in graphical format. As an example; the operation of resetting a photodiode contains only the movement of a few switches in a timing diagram generated by an RTL simulator. But when viewed as such, it is very difficult to analyze and debug, in particular when one also has to be able to understand the value entered in a line pointer from the same timing diagram. A block in a flow chart called “reset photodiode” next to “pointer value increase” is on the other hand easy.

In this document, there will be a flow chart example shown, that users may inspire from while planning code for the STAR and HAS sensors.

The Verification and Debugging Phases

Even when following good coding practices, it is unlikely that the sequencer will operate completely correctly without some debugging, but the debugging should actually be preceded by a code *verification* phase and associated documentation.

Code verification involves checking that the output of the code – in this case simply the switching of a few control signals – meets the expected behavior. This work should ***not*** be done by investigating unprocessed timing diagrams from the simulator output. Instead, the work must be divided up by resusing the block diagram from the code planning phase, so that the output from the code compared to the specifications in each block step by step.

Similarly, the behavior of the line pointers described below may be checked against the expected behavior by simply plotting their values on a time axis. This will catch one common mistake seen among customers, where the first and last lines of the image get different integration times due to unexpected counting of the dummy lines.

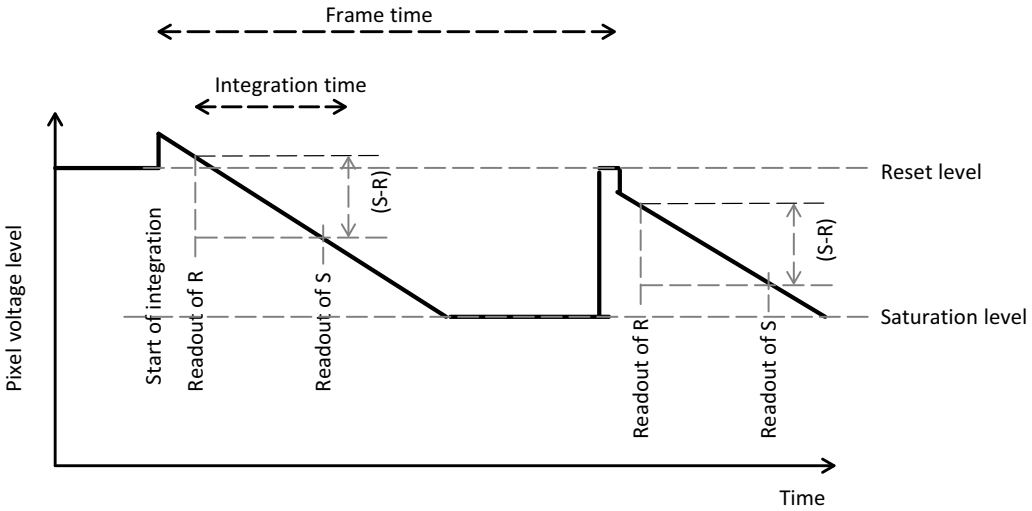


Figure 1. Principle of Correlated Double Sampling Eliminating kTC−noise

Basic Concepts

Photodiode and Read−out

Pixel structure and (S−R) subtraction

The pixel in the STAR and HAS sensor consists of one photodiode and a few transistors (Figure 2). A readout of a pixel value normally consists of three steps:

1. The signal S is sampled on the OUT bus
2. The photodiode is reset
3. The reset signal R is sampled on the OUT bus

Outside the pixel, a differential amplifier will transmit (S−R) the output stages. This type of operation removes static offsets between different pixels in the array and is called *uncorrelated* double sampling. Note that uncorrelated double sampling only reduces fixed noise sources, not temporal noise. Uncorrelated double sampling is hence done as a way of reducing Fixed Patter Noise (FPN).

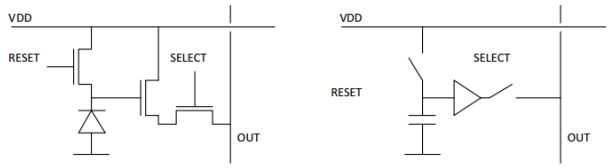


Figure 2. Pixel Structure (left). Equivalent Circuit (right).

Temporal Noise in Low Illumination

kTC noise

The photodiode has a certain inherent capacitance, which means that every reset will induce some kTC noise, which will become the temporal noise that dominates in low illumination. This happens because the reset level of a pixel is sampled as the R−value for every frame, while the actual reset level in a frame is varying from frame− to frame, as illustrated by Figure 3.

There are two methods of mitigating this temporal noise:

1. Softening of the reset. This means to adjust the voltage at the gate of the reset transistor during the reset phase.
2. Use *correlated* double sampling instead of *un*correlated. Correlated double sampling is possible in devices that support non−destructive readout (e.g the HAS−2).

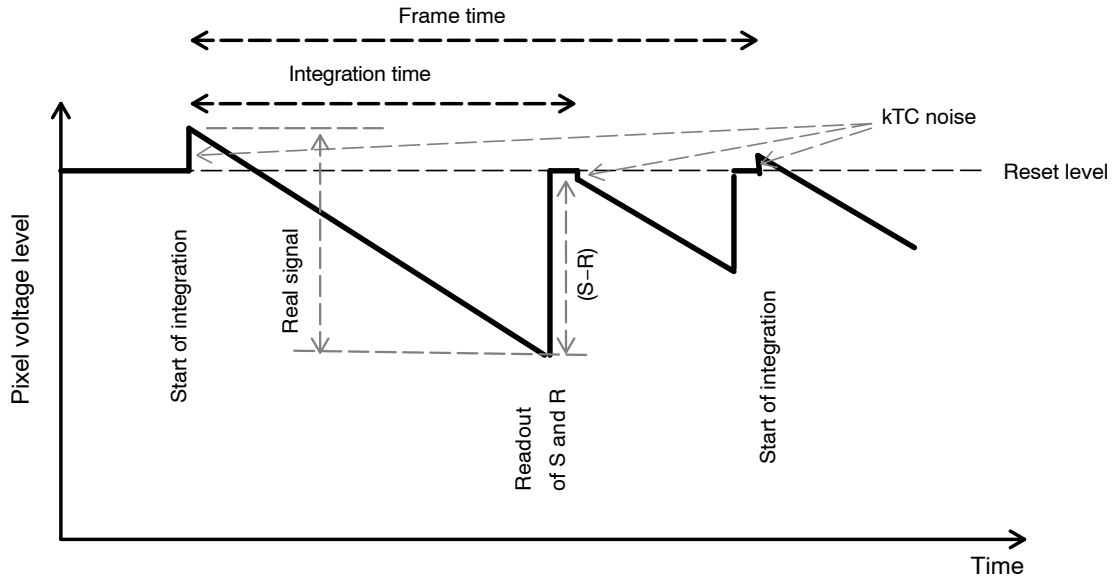


Figure 3. The Influence of kTC−noise during "Normal" Destructive Readout Operation

The real signal is the difference between the reset value including kTC noise at the beginning of integration, but the (S−R) that is read out is using the the R−value following a second reset operation and hence includes a different kTC noise.

Correlated Double Sampling and Non−destructive Readout

Correlated Double Sampling is possible when there is a function for reading out the sampled values after reset and after integration separately, so that these two values may be subtracted. The subtraction may in principle be done on the chip or outside the chip, but in the case of the HAS−2, only off−chip implementations are possible. On the STAR−devices, CDS is not possible.

In the case of the HAS−2 sensor, correlated double sampling (CDS) can be achieved by reading out the R−value of the pixel immediately after reset and storing it in an

R−frame externally. Note that this requires that the R−value is read out using a *fixed* voltage reference Vr that is common for all the pixels, i.e. the voltage on the output is (R−Vr), not (S−R) since no S is available. Hence, no FPN correction is done on the R−frame inside the chip.

After the integration time is over, the S−values of all pixels are read out as an S−frame, again without an FPN correcting reference. As the S−frame is being transferred to the FPGA, the values of the stored R−frame are used for the (S−R) subtraction in the digital domain, i.e. not on the chip. In the subtraction process, both FPN and kTC−noise is reduced.

Reading out S and R without resetting the diode is called "non−destructive readout".

Note that the read−out of the R−values and S−values may be done in different ways depending on the desired integration time, as will be explained in this document.

Rolling Shutter Operation

Figure 5 illustrates the basic rolling shutter operation with two line pointers P_{reset} and P_{read} moving downwards across the pixel array at a fixed line rate.

One pointer is used to indicate which line to reset, while the second is used to indicate which line to sample and read out. Hence, the photodiodes of line $\boldsymbol{i}$ are first reset when $P_{reset} = i$ then integrating until $P_{read} = i$.

As the pointers reach the last line at the bottom of the array, they may "roll over" − i.e. as the reset pointer reaches the bottom of the image, it may start over at the top again, thereby keeping a constant and non−interrupted line rate.

Since the pointers move with a fixed line time Δt_{line}, the distance between the pointers as measured in number of lines, $\Delta lines$, is also a measure of the integration time. Figure 4 illustrates this by showing the operations done per line in a sequential order. At time t_i, the line pointer for reset is pointing at line i, so $P_{reset} = i$. Once the pointers have moved 3 steps, the pointer for readout has reached line i, i.e. $P_{read} = i$, and the line is read out with an integration time of $3 \times \Delta t_{line}$,.

"Electronic Shutter"

The expression "electronic shutter" is an old expression that can have different meanings. In the case of rolling shutter sensors, it simply means that the integration time is determined by operations on two rows instead of one, as has already been illustrated in Figure 5. Very old rolling shutter sensors did not have this feature and hence, the integration time was then always fixed to be the same as the frame time.

The Blanking Time

As illustrated in Figure 4, the integration time of a line is ended by the line operations required to get the data out. The first part of the line operations is the blanking time, during which no pixels are read out, but important signal transfers are happening inside the sensor. During the blanking time, the signals from the photodiodes in line P_{read} are transferred to the readout stages in the sample operation and the photodiodes of the line that P_{reset} points to is reset.

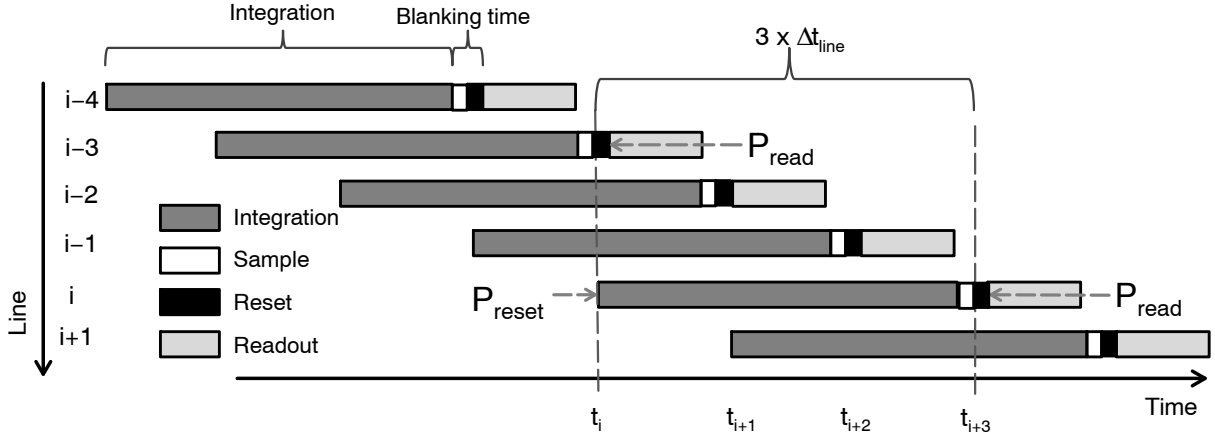


Figure 4. Rolling Shutter Operations Illustrated in Line−per−line Form

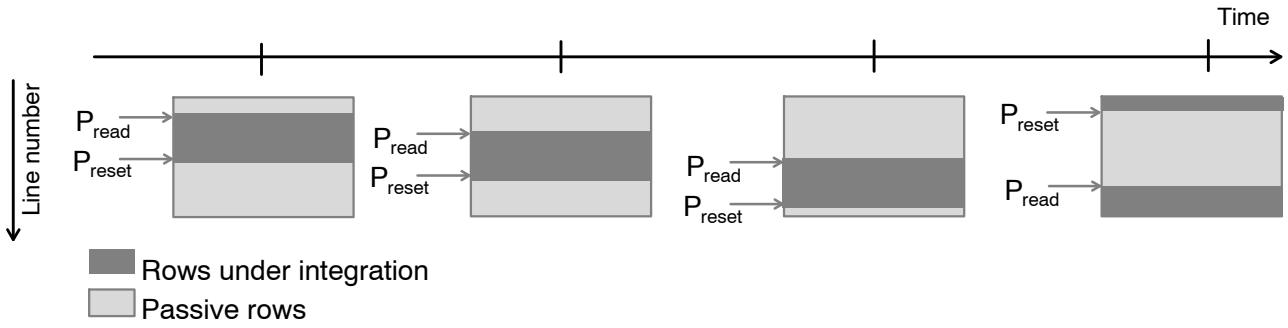


Figure 5. Illustration of Rolling Shutter Principle

Dummy Lines

The maximum integration time is not limited to the number of lines times Δt_{line}, . It is possible to extend it in different ways, but a common method is to use “dummy lines” that the pointers may point to, but have no physical implementation.

Reading out Pixels with ROI

Once the pixel values are sampled on the readout stages, they need to be clocked out of the sensor. In the STAR and HAS2 sensors, this can be done in a windowed mode implemented by a start pointer for the pixel readout sequence, which will be here written as X_{start}.

This parameter defines the first pixel to be read out from the line and the pixels with smaller index than X_{start}. will not be. read out. The number of pixels to be read out from a line needs to be controlled from the external sequencer by simply ending the readout operation when the intended length of the ROI has been read.

Active Register Selection

Operations on the lines pointed at by P_{reset} and P_{read} may not be done in parallel in the STAR250 and HAS−2 and a Boolean input for selecting which of the two registers to activate is provided.

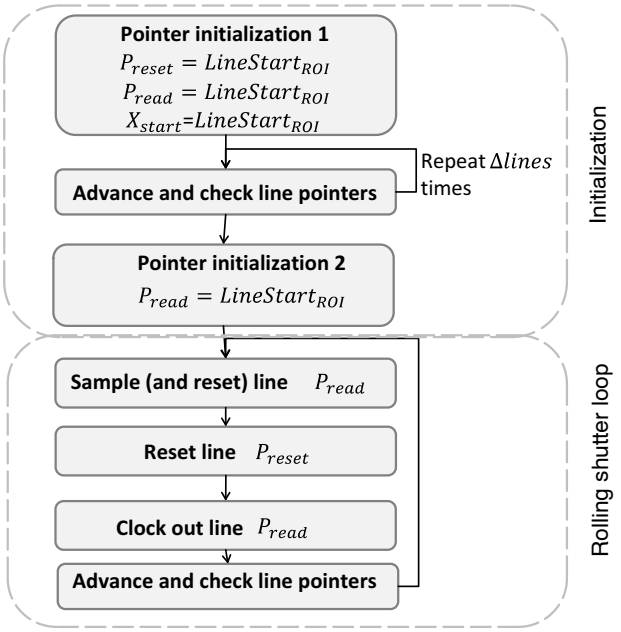
Simplified Generic Rolling Shutter Sequence

Introduction

Since all the rolling shutter sensors work according to the same principles, it is possible to define a generic rolling shutter sequence that can be easily implemented on the STAR and HAS devices. One example for such a sequence will be shown here for the simplified case of a single ROI readout. It is based on only a few input parameters:

$LineStart_{ROI}$, $LineEnd_{ROI}$, $ColStart_{ROI}$ and $ColEnd_{ROI}$ define the outer limits of the ROI to be read out.

$\Delta lines$ defines the integration time as the number of lines between the two line pointers.



Note that two lines are reset during a step of the rolling shutter loop: First the P.read line is reset to provide the R−value for the readout. Secondly, the line P.reset is reset in order to control the integration time.

Figure 6. Simplified Generic Sequence for Rolling Shutter Operation with Destructive Readout

Block Diagram

The sequence shown in Figure 6 begins with a pointer initialization step, in which both line pointers and X_{start} are programmed. After this, the line pointers are advanced Δ*lines* steps forward before the P_{read} pointer is once again programmed to the top of the ROI. At this stage, the line pointers have values differing by Δ*lines*, the P_{read} pointer is directed to the first line of the ROI and all lines in between have been reset. Hence, the rolling shutter loop may be entered.

While exact implementations may vary, the **type** of flowchart illustrated by Figure 6 should be implemented for a “debuggable” implementation of a sequencer.

The Advance and Check Line Pointers Block

It will be shown below that most of the blocks in Figure 6 become trivial to implement based on information from the data sheets. The Advance Pointers block is an exception, since a few implementation−specific aspects must be considered here:

Triggering of the advancement

The advancement of the pointers can either happen immediately after the end of the pixel clock out, or it can be configured to wait for an external trigger that controls the line rate outside of the sequencer.

Mirror representation of the pointer values in the FPGA

The line pointer values are stored in registers on the sensor for the HAS−2 and STAR−250, and their values can be incremented with simple clocking of the registers. It is hence tempting to skip storing the values of the line pointers in the FPGA, since they are stored and incremented on the sensor.

It is however advised to keep some form of synchronized mirror registers of the pointers in the FPGA, as these can be used for pixel/line sorting and end−of−ROI detection.

End of ROI actions

As the line pointers reach the bottom of the ROI, they need to be reset to the top of the ROI again. The sequencer running on the FPGA is responsible for detecting the end of the ROI, as the STAR and HAS−2 sensors can not do it on−chip.

Dummy lines

In case integration time is corresponding to more lines than the sensor has are intended, then the best place to handle this is inside the advance pointers block. Different implementations can be used, but may for instance follow a procedure as:

- Create flags and counters for the two line pointers. The counters are to be used when the pointers are pointing at dummy lines, i.e. when they are in dummy mode. The flag is to indicate that the counter is in dummy mode.
- Once a pointer reaches the end of the array, the flag for dummy operations on that pointer is raised. Note that this is done individually for the two line pointers.
- The sample, reset and clock out operations check the flag and use it as a gate. If it is raised for the read−pointer, then all operations regarding sampling and clock−out are gated. If it is raised for the reset pointer, then the reset operation is gated. Gating here means that the same time is spent on the operation, but no outputs are switched.
- The advance line pointers operation will instead of increasing the value of the pointer increase the counters when they are in the dummy mode. Once the counter reaches the intended number of dummy lines, the pointer leaves dummy mode, the dummy operation flag is deasserted and the pointer is directed to the top of the frame.

Other implementations for dummy lines are imaginable.

NDR/CDS Read−out Sequences

Simplified Case: Full Frame Operation

Figure 7 shows the simplest case of using the sensors in CDS mode. The sequence may be described as:

1. The full frame (or ROI) is reset without reading out any pixel data. This operation can be done quickly.
2. The R−frame is read out
3. The sensor is left integrating for a certain integration time
4. The S−frame is read out.

After this, the system returns back to the first step and resets the full ROI once again. Note that there is no “rolling shutter” loop in this mode of operation.

Interleaved Case for CDS/NDR with Short Integration Times

There is also a possibility of running the HAS−2 sensor in CDS/NDR mode with shorter integration times than the full frame operation case allows.

As shown in Figure 8, it does however require that the values from the R−frame and the values from the S−frame are sampled in an interleaved sequence, which is more complicated to handle on the receiving side.

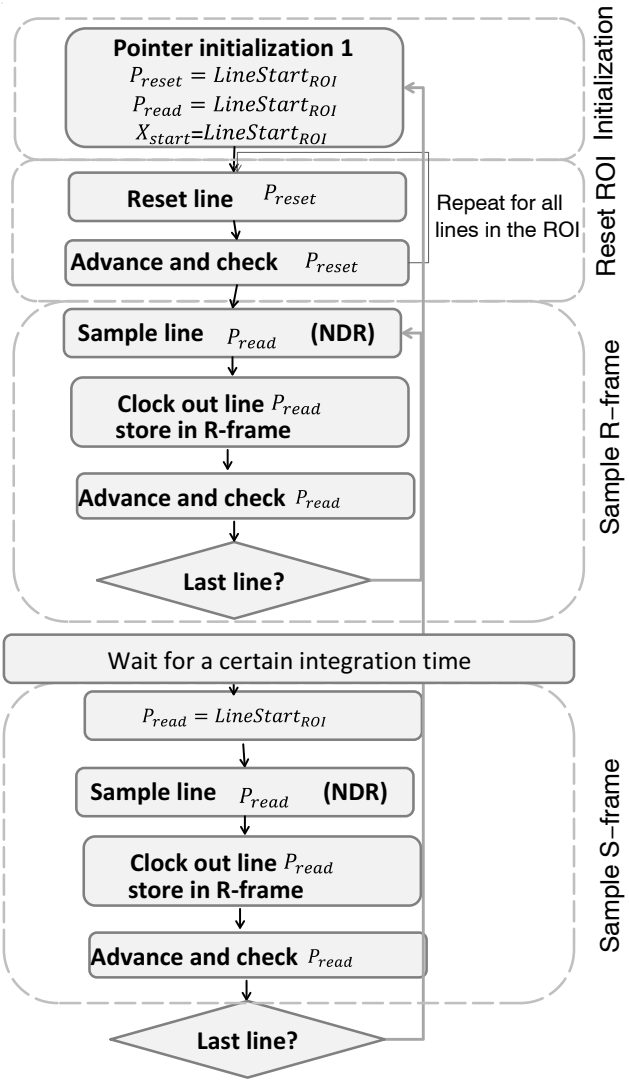


Figure 7. Generic Sequence for NDR/CDS Operation with Full Frame Readout

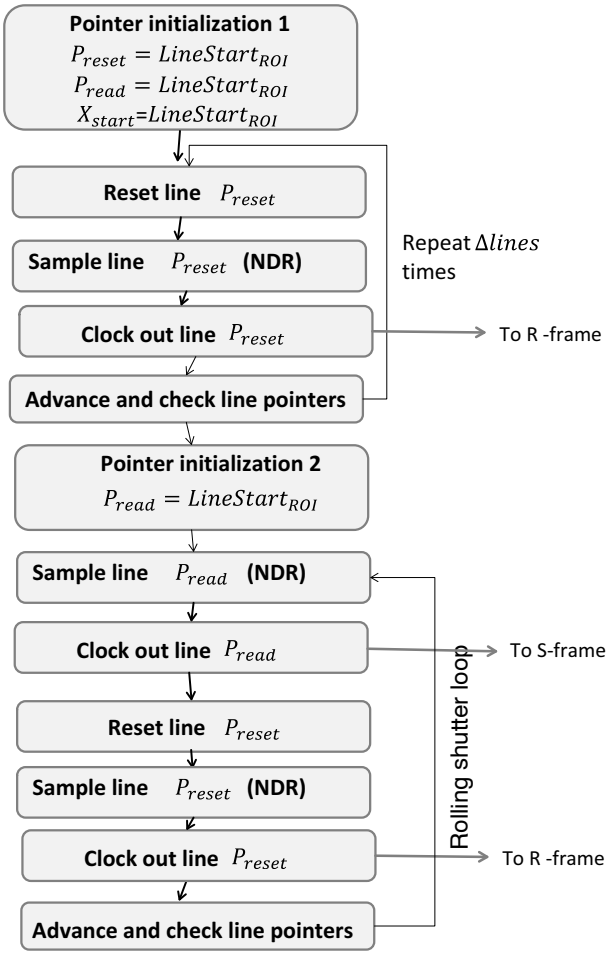


Figure 8. Sequence for Interleaved CDS/NDR Operation

HAS2 Specifics

Initializing Pointers

Unlike the STAR sensors, the HAS−2 has dedicated and appropriately named pointer registers for P_{read} and P_{reset}, namely the YRD and YRST registers. In order to program these registers to a specific value, one must first load the value into a third register Y1, which is done in a two−step process:

- The value on the input bus A[9:0] is transferred to Y1 by strobing the LD_Y input.
- The SYNC_YRD or SYNC_YRST inputs are triggered to transfer the value in Y1 to either YRD or YRST.

The loaded value in Y1 will be retained and may be reused.

The X1 register and the LD_X input provides the same function for X_{start}.

Figure 32 of the data sheet provides information on the constraints for loading and clocking the Y−registers. This figure is reproduced in this document as Figure 9 with a few comments.

Note that a simplifying implementation is to tie the CLK_YRD and CLK_YRST clocks together into a common CLK_Y input, all required operations are still possible in such a configuration.

Incrementing Pointers

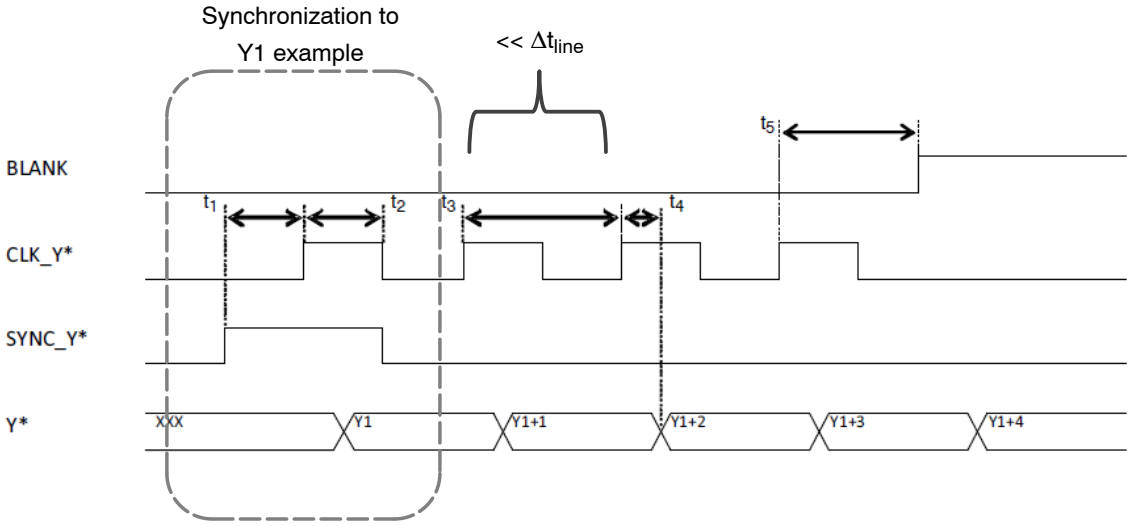
Both YRD and YRST are fully implemented shift registers. There are two clocks for the line pointer registers in the HAS−2 device, CLK_YRD and CLK_YRST, so they may be advanced independently. The registers are saturating, i.e. when reaching their maximum value, they will not roll over to 0. Hence, the sequencer needs to check if a register has reached its maximum and re−sync to Y1 if this has happened.

Active Register Selection

In the HAS−2, the input **YRST_YRDn** selects either YRD or YRST as the active register for operations. This input ***only*** needs to be controlled in the sampling and reset operations, as the shift register have individual input clocks and sync functions.

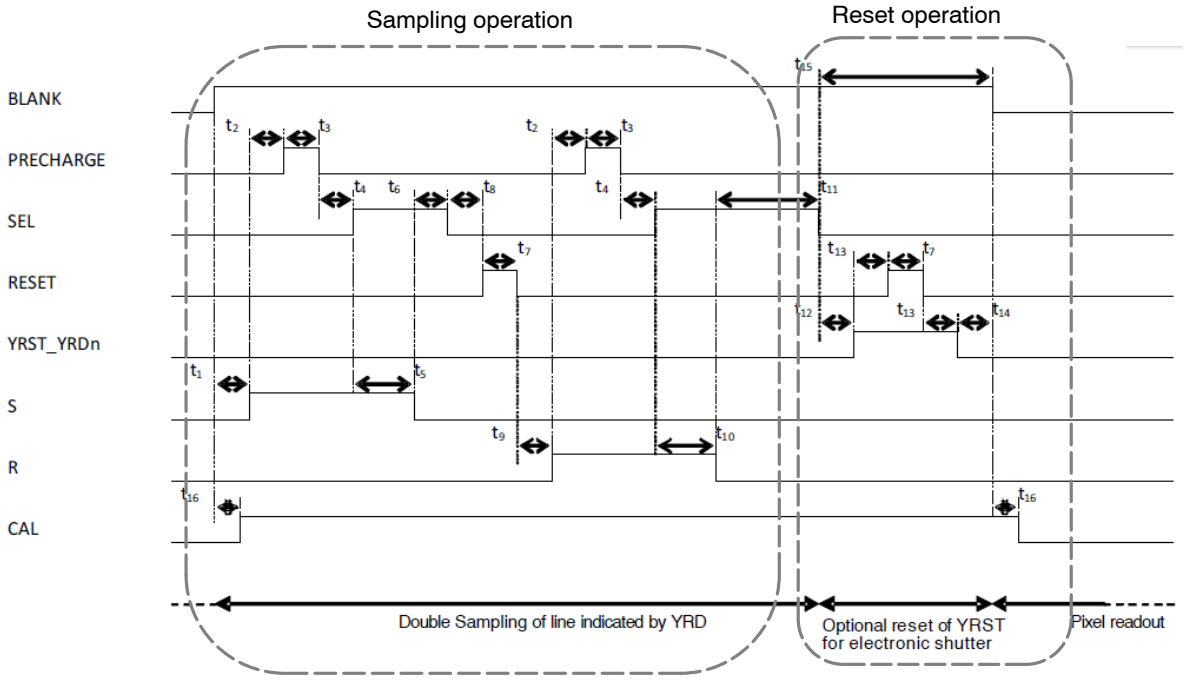
The Sampling and Reset Operations

The sampling and reset operations are described in Figure 34 of the data sheet and reproduced with some comments in Figure 10 in this document. Note that YRST_YRDn needs to be raised during the reset operation, so that the YRST register becomes active and the correct line is reset.



This figure is intended to show only the timing tolerances of the line pointer register clocking − it does not show the sensor under normal operation, since the blanking time occurs once per line.

Figure 9. Figure 32 from the HAS−2 Data Sheet Reproduced with a few Comments



Note that the “Sampling operation” indicated above resets the photodiode after the S−value has been sampled, so that and R value may be provided for uncorrelated double sampling. Hence, each line operation contains two resets: One for the photodiode being read out, and a second for the photodiode on the reset−line.

Figure 10. Required Switches for the Sampling and Reset Operations as Described in the Data Sheet

Pixel Readout Operation

Similarly to the sample and reset operations, the pixel readout (or “clock out”) operation should be simply copied from the data sheet and meet the indicated timing tolerances.

EOS

The HAS−2 has an End−Of−Scan (EOS) monitor output that *can* be used to simplify code ***if and only if*** full frame operation and no dummy lines will be used. Note that the EOS has one output only and that a register setting is controlling which register (XRD, YRD or YRST) that has its saturation indicated on the output pin. Instructions for its use are given in the data sheet.

The CAL Input

The sensor has an input labelled CAL, which will set the offset of the output amplifier to a fixed value defined by the analog input blackref. The functionality is similar to an auto−zero procedure, where a reference value is stored on a capacitor at the reference input of the output amplifier after the CAL procedure. It is enough to raise CAL once per frame or once per line.

STAR 250 Specifics

Line Pointers

Initializing Line Pointers

The STAR 250 has two shift registers for the Y−pointers. These are simply named YL and YR, where L and R mean left and right.

In the sequences described in the data sheet, the left is used for keeping track of which register that is read out () and the right for keeping track of which line to reset () in case the integration time is shorter than the frame time (i.e. “electronic shutter”.)

The Y−start register

The STAR250 has a Y−start register that can be configured with the starting value of an ROI by using the LD_Y input. The Y−start can then be transferred to the YL and YR registers with the SYNC_YL and SYNC_YR inputs.

Incrementing line pointers

The line pointers of the STAR 250 provided can be incremented by *either*:

- Clocking the shift registers. Since the two shift registers YL and YR have independent clock inputs, it is possible to advance the line pointers independently, *or*
- Continuously uploading new values with LDY and synchronizing them to the shift registers with SYNC_YL and SYNC_YR.

In case of the first alternative, one may keep the starting value of the ROI in the Y−start register and use the SYNC_YL input to reinitialize the line pointer in YL register when YL reaches the end of the ROI (and vv for the YR register)

Saturation of line pointer registers

The line pointer−registers YR and YL are saturating, i.e they will not “roll over” when they reach their maximum value and must be reinitialized for every new frame. Note that reinitialization must be done separately for YL and YR, since they are normally having different values and will hence normally saturate at different times.

The EOSYL and EOS_YL outputs will flag when saturation occurs, but it is by no means required to use these flags to control the sensor, since the values of the shift registers can be mirrored outside the sensor as well.

Active Y register selection

The L/R input of the STAR 250 determines which line that the RESET, SEL, S or R operations will act on.

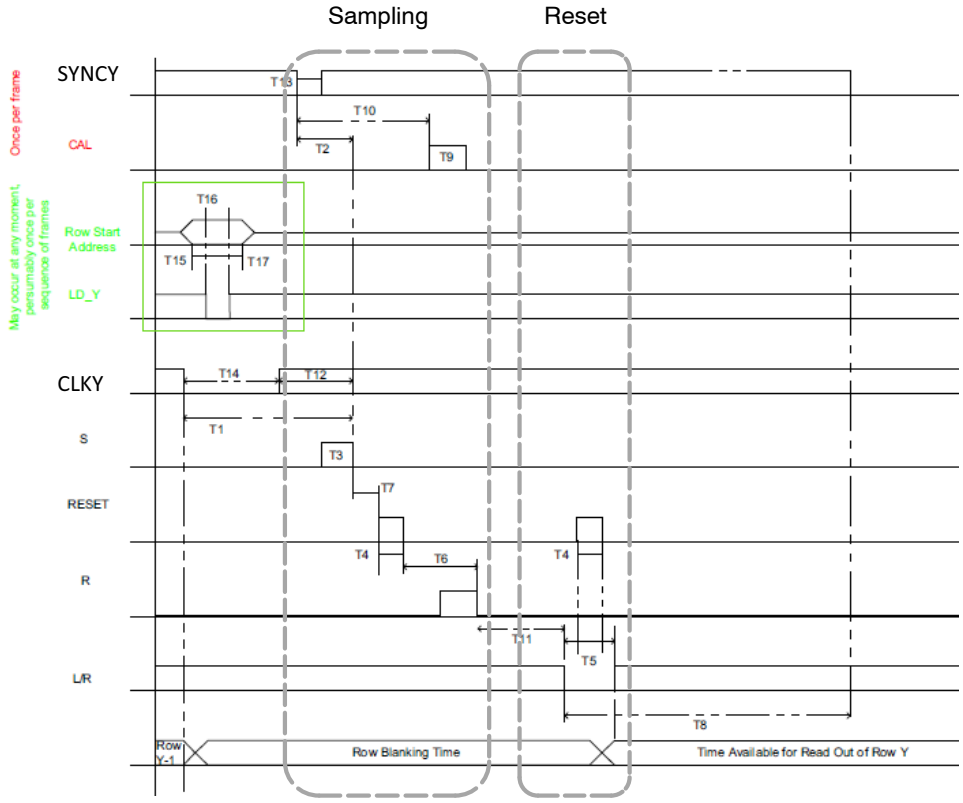
Sampling and Reset Operation

The basic sampling and reset operations are illustrated in Figure 11. Note there are two reset occurring per line. It is for this sensor *recommended* to do the reset of the pixels in the reset−line () after the read−line () has been sampled. The L/R input is toggled for the reset of line .

The SYNC_Y and CLK_Y lines shown in the top of Figure 11 are included to indicate the limits T2, T12, T13 and T14. Exactly which line pointer functions that will be done before the sample and reset operations will vary depending on the implementation and will not necessarily be the same for all lines.

The CAL Input

The sensor has an input labelled CAL, which will set the offset of the output amplifier to a fixed value defined by the analog input blackref. The functionality is similar to an auto−zero procedure, where a reference value is stored on a capacitor at the reference input of the output amplifier after the CAL procedure. It is enough to raise CAL once per frame.



Note that the “Sampling operation” indicated above resets the photodiode after the S−value has been sampled, so that and R value may be provided for uncorrelated double sampling. Hence, each line operation contains two resets: One for the photodiode being read out, and a second for the photodiode on the reset−row.

Figure 11. Figure 11 from the STAR 250 Data Sheet Reproduced with some Comments

STAR 1000 Specifics

No Line Pointers!

The STAR 1000 is significantly different from the HAS2 and STAR 250 in the sense that the STAR1000 has no shift registers to increment X− and Y− pointers on the chip. Instead, there is a 10b address input bus that is used for loading new X and Y pointer values every time the value of a pointer should change.

There are however two registers to store the pointer values, one for X and one for Y. These registers are loaded with the values on the address bus by toggling the LD_Y and LD_X inputs.

The CLK_X Input

As the STAR1000 lacks shift registers, there is nothing to be clocked by the CLK_X input. Instead, this input has a similar functionality as a SYNC – input, i.e. it will trigger the decoding of what is on the X−register to the column pointer. Hence, the CLK_X input triggers the column selection.

The CLK_X also has *one more function*: It triggers the output stages to perform the (S−R) subtraction operation and to connect the result to the output amplifier. As indicated in the data sheet, one may see this as a delay counted in toggles on CLK_X between the time that a column is selected and the time the pixel value from that column is appearing on the output.

The same delay also has some important effects when reading out the *last* columns of a line: The CLK_X needs to toggle even when there is no new column address to load, as the end of the line is reached.

Sampling and Reset Operation

Figure 8 from the data sheet is reproduced with some comments in Figure 12 below.

The CAL Input

Similarly to the HAS2 and STAR250, the STAR1000 has an input labelled CAL, which will set the offset of the output amplifier to a fixed value defined by the analog input blackref. The functionality is similar to an auto−zero procedure, where a reference value is stored on a capacitor at the reference input of the output amplifier after the CAL procedure. It is enough to raise CAL once per frame.

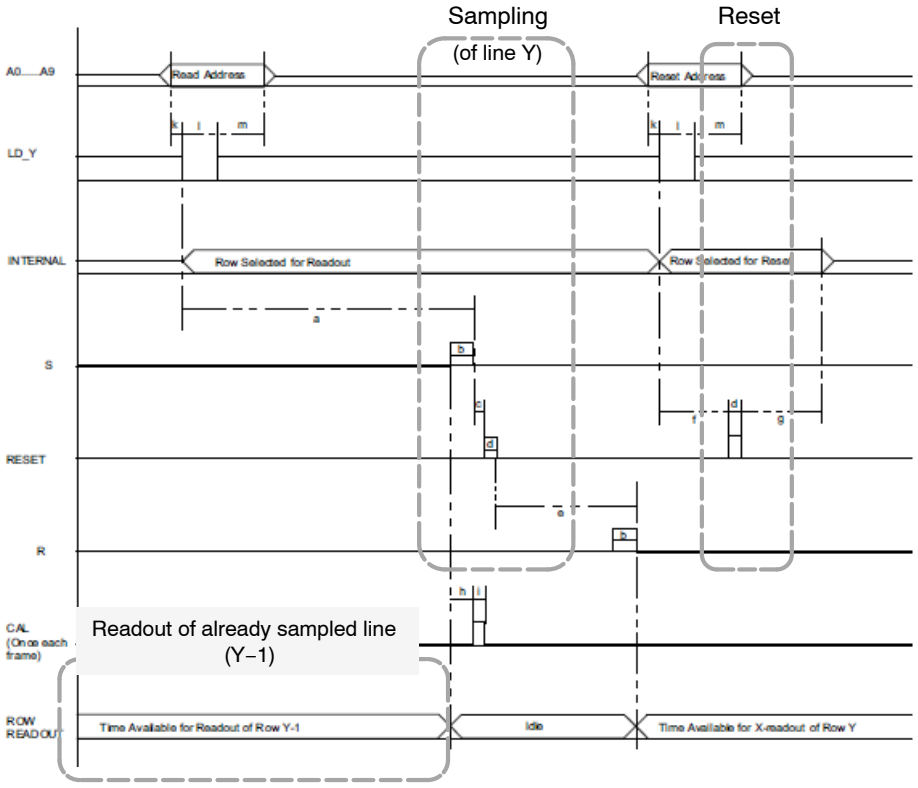


Figure 12. Figure 8 from the STAR 1000 Data Sheet Reproduced with some Comments

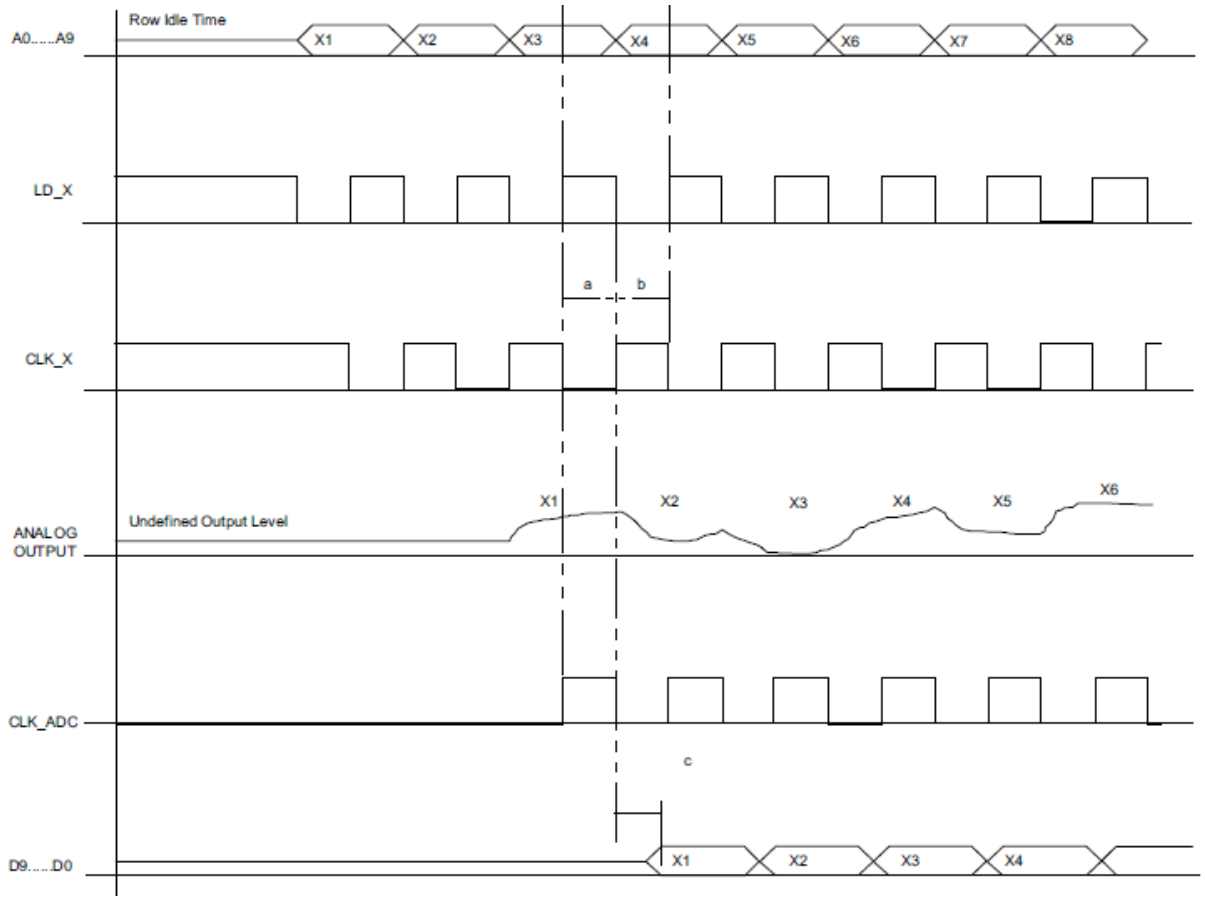


Figure 13. Figure 9 from the STAR 1000 Data Sheet Reproduced with some Comments

Other Notes

Multiple ROIs in STAR and HAS2 Devices

Some advanced image sensors with on−chip sequencers allow for multiple ROI functionality. As an example, one may mention the VITA and Python families. In such devices, one configures a set of ROIs by uploading coordinates, and the sensor will automatically output only pixels within the ROIs.

It is not possible to do this directly with the STAR and HAS2 devices, because they do not have on−chip sequencers that deliver the pulses for pixel operation etc (which is why this document exists). But multiple ROI operation is still possible, because the pixel addressing is also done from the outside, so the same operations that an advanced on−chip sequencer is doing in the case of a VITA sensor, may be done by the radiation hard FPGA in the case of the STAR and HAS2 sensor.

Hence, in the case of the STAR and HAS2 devices, multiple ROI operation can be done, but it is primarily an exercise for the FPGA programmer to continuously configure the line and pixel registers during operation, so that only the relevant pixels are output. I.e. the same job that is done by an on−chip sequencer needs to be done by an off−chip sequencer.

Abbreviations

NDR	Non destructive Readout
CDS	Correlated Double Sampling

ADDITIONAL INFORMATION

TECHNICAL PUBLICATIONS:
Technical Library: www.onsemi.com/design/resources/technical−documentation
onsemi Website: www.onsemi.com

ONLINE SUPPORT: www.onsemi.com/support
For additional information, please contact your local Sales Representative at www.onsemi.com/support/sales