Getting Started

Getting Started guide for the FLIR ADK with USB, GMSL, and Ethernet Connector Options

FLIR Systems OEM & Emerging
6769 Hollister Avenue
Goleta, CA 93117
Phone: +1.805.964.9797
www.flir.com

Document Number: 102-2013-105
Version: 130
Issue Date: 2019

The information contained herein does not contain technology as defined by EAR,15 CFR772, is publicly available, and therefore not subject to EAR.

Table of Contents

The information contained herein does not contain technology as
defined by EAR,15 CFR772, is publicly available, and therefore
not subject to EAR.

1 Introduction

1.1 Revision History

Version	Date	Comments
100	05/02/2019	Initial Release
110	7/9/2019	16-bit Video + cable spec
120	10/14/2019	Ethernet Description
130	11/15/2019	GMSL Description + USB Description

1.2 Reference Documents

Ref Number	Document number	Comments
1	102-2013-40	Boson datasheet

1.3 SCOPE

This document describes detailed instructions on the conditions and requirements for integrating the FLIR ADK with various communications methods including USB, Ethernet, GMSL accessory board for communication, video streaming, and hardware synchronization.

1.4 BACKGROUND

The FLIR ADK comes in three variants USB output, GMSL output, and GMSL output with a converter for GMSL to Ethernet. This document describes how to get each version up and imaging.

2 USB – Getting Started

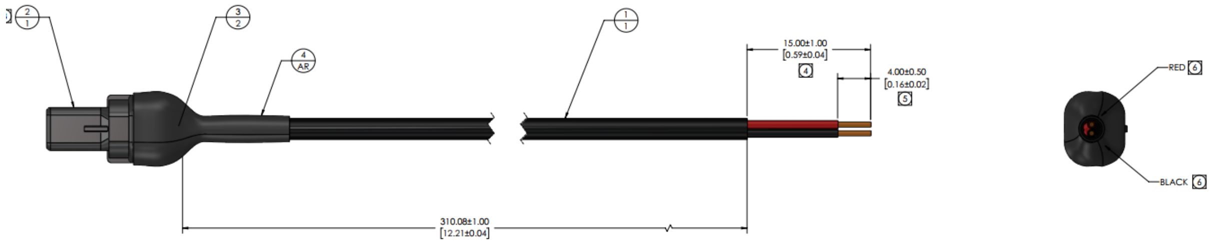
The USB option for the ADK ships with a Boson connected by USB for video and command and control, a BNC cable for external sync input, and a Mizu-P25 connector to power the window heater.

2.1 Connect and power the USB ADK

1) Connect the USB cable to the computer.
2) Optional: To power the window heater, apply 12 VDC to the MIZU-P25 connector (shown below) to power the heater. The heater is protected against reverse polarity so it's technically ok to hook it up in reverse, it will just pull a little more power. The red lead

The information contained herein does not contain technology as defined by EAR,15 CFR772, is publicly available, and therefore not subject to EAR.

shown in the drawing below is the positive lead to the heater.



2.2 Streaming Video on Windows

The camera controller software can be found here

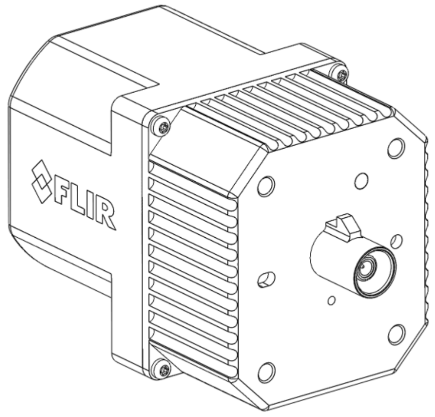
https://www.flir.com/support-center/oem/camera-controller-gui-for-boson/

After the software is installed find the camera comport by opening Device Manager and listing the available ports. Do this with the camera unplugged and with the camera plugged into the computer. The port that shows up with the camera plugged in is the camera's comport.

With the camera's comport known, open the BosonGUI software and select the camera's comport in the Port selection dropdown in the lower right corner of the BosonGUI software. To start video streaming just click the start video capture button in the upper right corner of the BosonGUI software.

3 Ethernet – Getting Started

The Ethernet option of the FLIR ADK ships with a GMSL connector on the camera and a GMSL to Ethernet Bridge that allows users to interact with the FLIR ADK using the standard GenICam protocol.



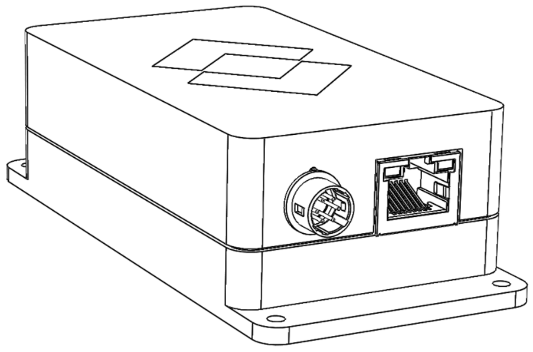


Figure 1: The FLIR ADK GMSL option is shown on the left with the GMSL 1 to Ethernet bridge shown on the right.

The information contained herein does not contain technology as defined by EAR,15 CFR772, is publicly available, and therefore not subject to EAR.

Getting Started

3.1 Connecting the Boson ethernet ADK to the computer

1. Connect the FAKRA cable from the FLIR ADK to the GMSL – Ethernet bridge.
2. Connect the ethernet cable to a network interface card on the host computer.
3. Connect the 6 pin GPIO cable to the GMSL Ethernet bridge.
4. Apply power to the 6 pin GPIO cable. Apply 8-16 VDC to the Green wire and ground on the Brown wire. The power supply should be rated for a 2 A peak current. The system is designed to run off the 12V supply from a car battery.

Table 1: Pin out of the GPIO pins on the Ethernet breakout box. The Colors correspond to the GPIO cable that came with the ADK.

Color	Pin	Function	Description
Green	1	VExt	Camera Input Power
Black	2	Not Connected	-
Red	3	Not Connected	-
White	4	Not Connected	-
Blue	5	Not Connected	-
Brown	6	GND	Camera Power Ground

3.2 Ubuntu Installation

We've tested the installation on Ubuntu 16.04.

1. Download the appropriate version of Spinnaker SDK from

   https://www.flir.com/products/spinnaker-sdk/

2. Unpack the tar.gz file for the appropriate system architecture.
3. Follow the README directions to install **necessary dependencies** and then run the **installer script**.
4. It's necessary to configure the ethernet card that the Boson is plugged into. It might be necessary to find which ethernet card the Boson is plugged into. To do this run 'ifconfig' with the ethernet cable unplugged then reconnect the ethernet cable and run 'ifconfig' again. You should see a change in one of the ethernet cards.

The information contained herein does not contain technology as defined by EAR,15 CFR772, is publicly available, and therefore not subject to EAR.

5. Now configure that ethernet card by opening the network interfaces file with your favorite text editor.

```
sudo gedit /etc/network/interfaces
```

Add the following lines (make sure to change enp15s0 to the appropriate card name found by running ifconfig)

```
iface enp15s0 inet static

address 169.254.0.64

netmask 255.255.0.0

mtu 9000

auto enp15s0
```

Lastly restart the networking service to apply settings

```
sudo /etc/init.d/networking restart
```

6. The installer installs a program called SpinView. This allows you to see live feed video from the camera. To start SpinView enter ‘spinview’ into the terminal.
7. When SpinView is open the Boson device should be visible under the devices tab (as shown below). Double click on the Boson device and the program will display a live feed from the camera.

If you notice that some images are getting dropped or that the video feed is choppy it may help to increase the Rx and Tx buffer size. Below is a link describing how to do that.

https://www.itechlounge.net/2015/05/linux-how-to-tune-up-receive-tx-and-transmit-rx-buffers-on-network-interface/

3.2.1 Robot Operating System

We have an example project that uses the Ethernet ADK in the ROS framework present here.
https://github.com/flir/flir_adk_ethernet

Once the Boson is setup to run with Linux then follow the instructions in the Readme.md to get the Boson to work with ROS.

After setup you should be able to run which will stream video on the host computer.

The information contained herein does not contain technology as defined by EAR,15 CFR772, is publicly available, and therefore not subject to EAR.

```
        roslaunch flir_adk_ethernet boson.launch
```

3.3 Windows Installation

Download and install the Windows version of the Spinnaker SDK here.

https://www.flir.com/products/spinnaker-sdk/

The Spinnaker SDK also comes with a software application called SpinView which will display real time images from the Boson Camera.

4 GMSL – Getting Started

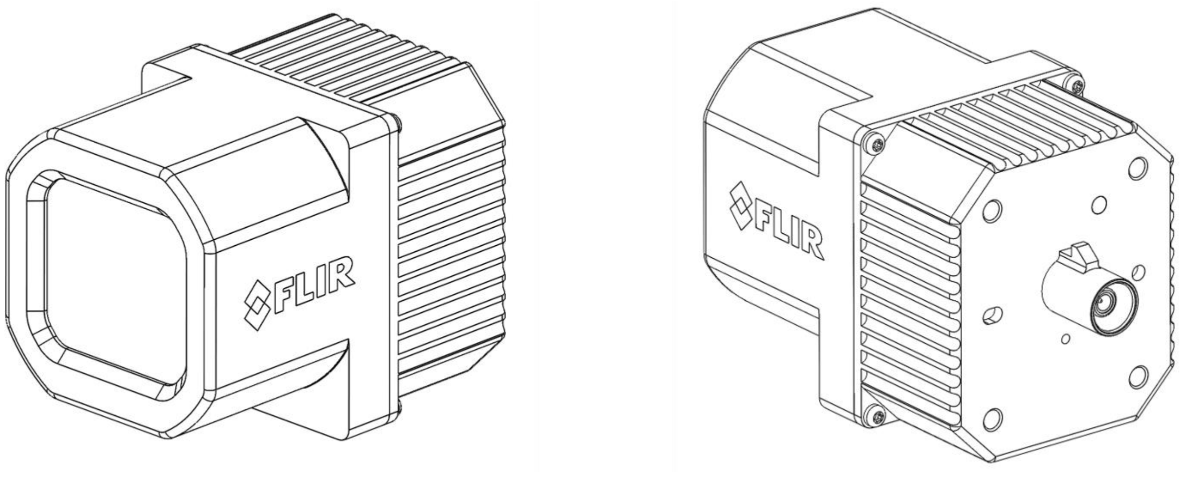


Figure 2: The FLIR ADK with the GMSL configuration.

The FLIR ADK with the GMSL connector allows for the longer cable lengths that are necessary in the automotive environment. The Serializer in the ADK can be configured for GMSL 1 (trailing part number AA1) and for GMSL 2 (trailing part number AA2). Note that Serializers configured for GMSL 1 can be field upgraded to GMSL 2.

In this document we provide examples of how to collect data from the GMSL ADK as well as read and write I2C commands to the ADK. We have a driver for the Nvidia Drive system and we show an example of using the Maxim MAX9296A_CSI_EVKIT to control the ADK. Note that at the time of writing this GMSL 2 is still Maxim proprietary information, thus it is necessary to contact Maxim directly to get documentation about the SerDes chips.

The information contained herein does not contain technology as defined by EAR,15 CFR772, is publicly available, and therefore not subject to EAR.

4.1 NVIDIA Drive Systems

The FLIR ADK features a heater and a shutter that are necessary for getting the most out of the thermal core. The heater and shutter require additional power than what is typically provided by the GMSL connection on the NVIDIA host. FLIR provides an additional power injector circuit if your application requires it. FLIR can also provide schematics for implementing the additional power injection on the deserializer circuit.

Connect the FLIR ADK to a FAKRA connector on the host system with the provided Coax cable.

FLIR provides a driver for the NVIDIA drive systems that enables the use of the FLIR ADK with the NVIDIA stack. The FLIR driver can be downloaded here.

https://flir.box.com/s/0tnhfoai7q9dq0a4qvkl0phfpo9kl26z

Follow the directions in the README.md to install the driver and get started with video streaming.

If the NVIDIA system is setup for GMSL 2 the ADK will have to be field upgraded to GMSL 2 for our driver to work. To field upgrade the ADK to GMSL 2 see section 4.2. If you know before hand that your system will require GMSL 2 cameras it is possible to order the ADK configured for GMSL 2.

4.2 Upgrading the GMSL Serializer to GMSL 2

The FLIR ADKs are by shipped in either GMSL1 or GMSL 2 mode from the. The GMSL 1 cameras have the trailing part number AA1 or AAX and GMSL 2 cameras have the trailing part number AA2. If the camera is in GMSL 1 mode and to be integrated on a GMSL 2 system it will be necessary to configure the Serializer on the ADK to use GMSL 2. This section describes how to configure the serializer on the ADK for GMSL 2.

ADK Serializer Address: 0x84 (8-bit addressing)
Deserializer address is typically 0x90 (Depends on system / deserializer configuration)

1) Configure deserializer for GMSL1 mode (note: in step 4 if you're not able to read back the register from the serializer this is because the deserializer didn't get configured for GMSL 1, a work around to configuring the deserializer to GMSL 1 is to set the CFG pin level of CFG1 to 1. This configures the deserializer to come up in GMSL1 with HIM enabled, this matches the Serializer default configuration.)
   des: write val 0x1F to reg 0x06
2) Turn on local I2C ACK on deserializer (Faked ACK response from the serializer)
   des: write val 0x80 to reg 0xB0D

The information contained herein does not contain technology as defined by EAR,15 CFR772, is publicly available, and therefore not subject to EAR.

3) Turn on high immunity mode (HIM), configure HV_SRC on deserializer:
   des: write val 0xE8 to reg 0xB06
4) Turn on CLINK on serializer side (forward channel enabled)
   ser: write val 0x43 to reg 0x404
5) Turn off deserializer local I2C ACK (because we don't need it anymore because forward channel from serializer is now working)
   des: write val 0x0 to reg 0xB0D
6) Turn on Parallel video mode on serializer (may be different for MIPI) (configures forward channel to be run from Boson's PCLK)
   ser: write val 0xF7 to reg 0x07
7) Switch from CLINK mode to SEREN mode on serializer (Now running off Boson's PCLK)
   ser: write val 0x83 to reg 0x404
8) Change from GMSL1 to GMSL2 mode on serializer (we lose communication to the serializer after doing this until deserializer is converted to GMSL2 as well) (upgrade remote side first)
   ser: write val 0x91 to reg 0x06
9) Change from GMSL1 to GMSL2 mode on deserializer (LOCK will be indicated after the link comes back up)
   des: write val 0xDF to reg 0x06

Note: The GMSL 2 configuration on serializer will be lost on power cycle, if running a GMSL 1 camera in a GMSL 2 configuration you need to implement this process each time.

4.3 GMSL SerDes Data Transmission

The Boson core of the ADK is wired to the Maxim Serializer as shown in Table 2.

Table 2: The Boson and Maxim 9295A pin correlation.

Boson	MAX9295A
Data bit 0	D2P, Pin 19
Data bit 1	D2N, Pin 20
Data bit 2	MFP5, Pin 21
Data bit 3	MFP6, Pin 22
Data bit 4	D3P, Pin 23
Data bit 5	D3N, Pin 24
Data bit 6	D0P, Pin 25
Data bit 7	D0N, Pin 26
Vsync	MFP3, Pin 17
Hsync	MFP4, Pin 18
PCLK	MFP0, Pin 2
External Sync	MFP7, Pin 31

The information contained herein does not contain technology as defined by EAR,15 CFR772, is publicly available, and therefore not subject to EAR.

4.4 Sending Commands to the ADK over i2C

Currently the Boson camera core of the ADK will only communicate via UART. The GMSL adapter board in the ADK has a i2C to UART FIFO transceiver that will allow you to send byte array commands to the Boson. In practice the I2C to UART buffer limits the amount of data you can send to the Boson at any one time to 128 bytes. The I2C to UART transceiver has the 8-bit address 0xD8.

The standard cycle for sending I2C commands to the boson is:

1) Turn off the transmitter on the I2C to UART transceiver.
    a. Write value 0x02 to register 0x09 to device 0xD8.
2) Send a series of I2C commands to the FIFO buffer.
    a. For each *value* in command byte array. Write *value* to register 0x00 to device 0xD8
3) Turn on the transmitter on the I2C to UART transceiver – this flushes the FIFO buffer to the Boson.
    a. Write value 0x02 to register 0x09 to device 0xD8.

Note: there are example i2cWrite, i2cRead, and sendCommand functions for a teensy board connected to the Maxim deserializer in Sample I2C Functions For Teensy SOC.

To generate the command byte arrays to send to the Boson, customers can use the rawBoson tool provided through FLIR on GitHub at https://github.com/FLIR/rawBoson combined with the Boson SDK documentation/Software IDD. The Software IDD describe the command the Boson will accept, and the rawBoson tool can packetize the message properly with a CRC code and start and stop flag. Documentation on how the CRC code and complete bytes need to be created and formatted will be provided at request. We recommend using this tool to determine the corresponding byte array to send for a desired command.

UART TX and RX FIFO are both 128 bytes, so large packets (like chunks of a camera firmware update) can overflow the FIFO if chunk size is not limited to ~64 bytes (in case the FIFO is not cleared between two successive large messages). Most if not all SDK UART commands do not exceed 64 bytes so this isn't a problem in practice except for file transfer. Note it's possible to update the camera firmware over GMSL, it will just take a while to send all the data over to the Boson.

Below is an example of turning the window heater on and off. Note that since the transmitter on the I2C to UART transceiver is not turned off these commands are written directly to the boson.

Turn Heater OFF – the heater is off by default

*Set GPIO state register on ADK side to configure heater **off***

The information contained herein does not contain technology as defined by EAR,15 CFR772, is publicly available, and therefore not subject to EAR.

*write val 0x0**1** to reg 0x19 to device reg 0xD8*
Configure heater GPIO as an OUTPUT (step can be skipped if already done this power cycle)
write val 0x0F to reg 0x18 to device reg 0xD8

Turn Heater ON

*Set GPIO state register on ADK side to configure heater **on***
*write val 0x0**9** to reg 0x19 of device reg 0xD8*
Configure heater GPIO as an OUTPUT
write val 0x0F to reg 0x18 of device reg 0xD8

4.5 Sample I2C Functions for Teensy SOC

Below are sample functions for i2cwrite and i2cread as well as a function to send an array of bytes.

I2C Read and Write:

```
void i2cwrite(byte addy, byte reg, byte val)
{
  byte err;

  Wire.beginTransmission(addy);
  Wire.write(reg);
  Wire.write(val);
  err = Wire.endTransmission();
  if (err != 0) {
    I2Cerror = 1;
  }
}

byte i2cread(byte addy, byte reg)
{
  byte err;
  byte res = 0;

  Wire.beginTransmission(addy);
  Wire.write(reg);
  err = Wire.endTransmission();
  if (err != 0) {
    I2Cerror = 1;
  }

  Wire.requestFrom((int) addy, (int) 1, false);

  while (Wire.available()) { // slave may send less than requested
    res = Wire.read(); // receive a byte
  }
  return res;          // send the byte
}
```

The information contained herein does not contain technology as defined by EAR,15 CFR772, is publicly available, and therefore not subject to EAR.

```
byte checkRXbuffer()
{
  //  byte addy = 0x6C;
  byte reg = 0x12;

  return i2cread(transAdd, reg);
  delay(50);
}

byte checkTXbuffer()
{
  //  byte addy = 0x6C;
  byte reg = 0x11;

  return i2cread(transAdd, reg);
}
```

Send I2C Command Array:

```
byte transAdd = 0x6C; // 0x6C - 7bit address. 0xD8 - I2C to UART Chip 8-bit
address


void sendI2CCommandArray(byte commandBytes[], int len)
{
  i2cwrite(transAdd, 0x09, 0x02); // disable the transmitter on transceiver

  for (int i = 0; i < len; i++){
    i2cwrite(transAdd, 0x00, commandBytes[i]); // write byte to fifo buffer
register on transceiver
  }

  if (debug_messages) {
    DEBUG.print("Message queued: 0x");
    DEBUG.print(checkTXbuffer(), HEX);
    DEBUG.print("\n");
    DEBUG.print("\nCommand Sent\n\n");
  }

  i2cwrite(transAdd, 0x09, 0x00); // enable the transmitter on transceiver.
This flushed the fifo buffer to the boson.
}
```

It's also necessary to write I2C commands to the Serializer and Deserializer that are 16-bit addressed. These are also shown here.

```
void i2cwrite16(byte addy, byte regH, byte regL, byte val)
{
  byte err;
```

The information contained herein does not contain technology as defined by EAR,15 CFR772, is publicly available, and therefore not subject to EAR.

```
  Wire.beginTransmission(addy);
  Wire.write(regH);
  Wire.write(regL);
  Wire.write(val);
  err = Wire.endTransmission();
  if (err != 0) {
    I2Cerror = 1;
  }
}

byte i2cread16(byte addy, byte regH, byte regL)
{
  byte err;
  byte res = 0;

  Wire.beginTransmission(addy);
  Wire.write(regH);
  Wire.write(regL);
  err = Wire.endTransmission();
  if (err != 0) {
    I2Cerror = 1;
  }

  Wire.requestFrom((int) addy, (int) 1, false);

  while (Wire.available()) { // slave may send less than requested
    res = Wire.read(); // receive a byte
  }
  return res;          // send the byte
}
```

4.6 Data Format and Boson Settings for GMSL 1

The GMSL option for Boson currently supports both 8-bit and 16-bit data. Of the 24-bit CMOS output from the Boson only the lower 8-bits are physically connected to the Serializer. 8-bit video over GMSL is available using CMOS configurable options that utilize the lower 8 bits (0-7) of the CMOS output. This is the standard 8-bit AGC video data output from the Boson.

For 16-bit video it is necessary to configure the CMOS output to multiplex the lower 8-bits of the CMOS output. To do this the CMOS channel must be put in IR16 video mode and then it must be set to Multiplex or “Double wide” mode.

4.7 Interpreting the Multiplexed/ Double wide IR16 video

The information contained herein does not contain technology as defined by EAR,15 CFR772, is publicly available, and therefore not subject to EAR.

The data comes across the GMSL link as a 1280 x 513 frame, where Column 1 contains the top 8 bits of Column 1 of the VGA sensor's first column, and Column 2 contains the bottom 8 bits of the VGA sensor's first column.

Stitching the top and bottom bits will yield the correct values for the IR16 image. The resultant frame should be the resolution of the sensor, 640x512, plus telemetry.

P1[15:8]	P1[7:0]	P2[15:8]	P2[7:0]	P3[15:8]	P3[7:0]	P4[15:8]	P4[7:0]

4.8 Syncing the Boson using GPIO pins of the SerDes

The Boson can be synchronized using the GPIO pins on the serializer and deserializer. For the MAX9295A serializer used on the ADK, one must correctly set MFP7 pin for D12 disabled. This includes doing a register write of 0x07 to register 0x0007 (It is originally set to 0xF7). This correctly works with the priority established by Maxim to enable GP0 over D12.

For GMSL 1, the deserializer will need to have an input into the GPI pin with a pulse width of >0.35us. The GPI pin on the deserializer will need to be linked to MFP7 on the serializer.

To establish sync with the Boson the Boson needs to be configured to One Shot mode and External Sync Slave mode. Note that sync pulses must be applied before the Boson is configured to External Sync Slave mode or the Boson will fault and subsequently restart. The Boson looks at the rising edge of the incoming sync signal, and pulse width has a wide range of accepted variability. For the GMSL deserializer the sync pulse must be a 1.8 V logic high pulse. Please see the Boson External Sync App Note Document for more information on the functionality of the sync feature with the Boson camera.

4.9 ADK GMSL Latency

The FLIR ADK has an associated latency between the time when a sync pulse is received and when MIPI data from the Deserializer reaches the SOC. This section shows the general latency stack up of the ADK.

Table 3: The latency stackup of the ADK GMSL option.

Data	Data Start	Data Finish	Time	Note
Ext Sync Pulse	Serializer	Deserializer	35 us (GMSL 1) 10 ns (GMSL 2)	
Ext Sync Pulse	Deserializer	Boson ROIC		The SOC on Boson has a direct feed through to the ROIC for frame acquisition start

The information contained herein does not contain technology as defined by EAR,15 CFR772, is publicly available, and therefore not subject to EAR.

Frame Acq	Boson ROIC Sync Received	Boson ROIC Frame Acq Start	288 us	
Frame Acq	Boson ROIC Frame Acq Start	Boson ROIC Frame Acq Complete	16.4 ms	The time it takes to read out a full frame.
Image Frame	Standard Image Processing 16-bit Start	Standard Image processing 16-bit complete	8.6 ms	NUC, Lagrange, DPR, SCF
Image Frame	Additional Image Processing (AGC) Start	Additional Image Processing (AGC) Complete	17 ms	Auto Gain Correction, Contrast Enhancement, Frame, conversion to 8-bit (*this step is optional)
Image Frame	Image Processing Complete	CMOS start of frame transmission	128 us	
Image Frame	CMOS - Serializer Start of frame transmission	Deserializer – Mipi Complete frame transmission	(16.67 ms) CMOS output frame transmission (~10 us) SerDes data transmission (depending on cable length.)	The time it takes from when CMOS output on the Boson starts and MIPI output from the Deserializer Completes frame data transmission

The GMSL signal pipeline introduces very little additional latency compared to the intrinsic latency of the Boson frame readout, image processing, and CMOS data output. The latency of the Boson core of the ADK is shown below for different image processing configurations. The latency reported here is defined as the time between when the Boson receives the external sync pulse and the time when the data transmission out of the CMOS output is complete. The lowest latency configuration is the 16-bit pre-AGC 60 Hz output shown in Figure 3 with a latency of ~42 ms. Turning the averager on to reduce the frame rate to 30 Hz increases the latency to ~59 ms, shown in Figure 4. Turning on the Auto Gain Correction algorithms and outputting 8-bit pixel data at 60 Hz has a latency of ~59 ms, shown in Figure 5. Turning on Averaging in the 8-bit pixel output increases the latency to ~75 ms, shown in Figure 6.

The information contained herein does not contain technology as defined by EAR,15 CFR772, is publicly available, and therefore not subject to EAR.

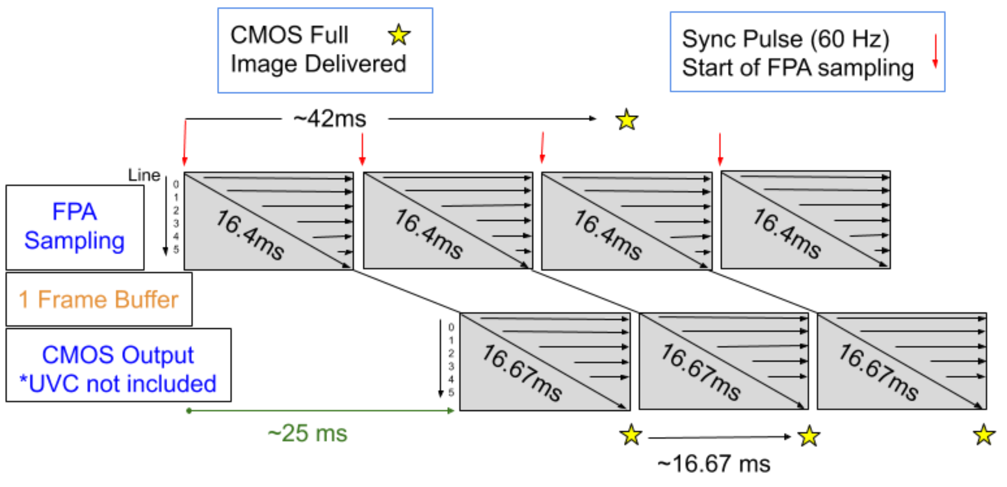


Figure 3: 16-bit pre-AGC CMOS Output 60 Hz

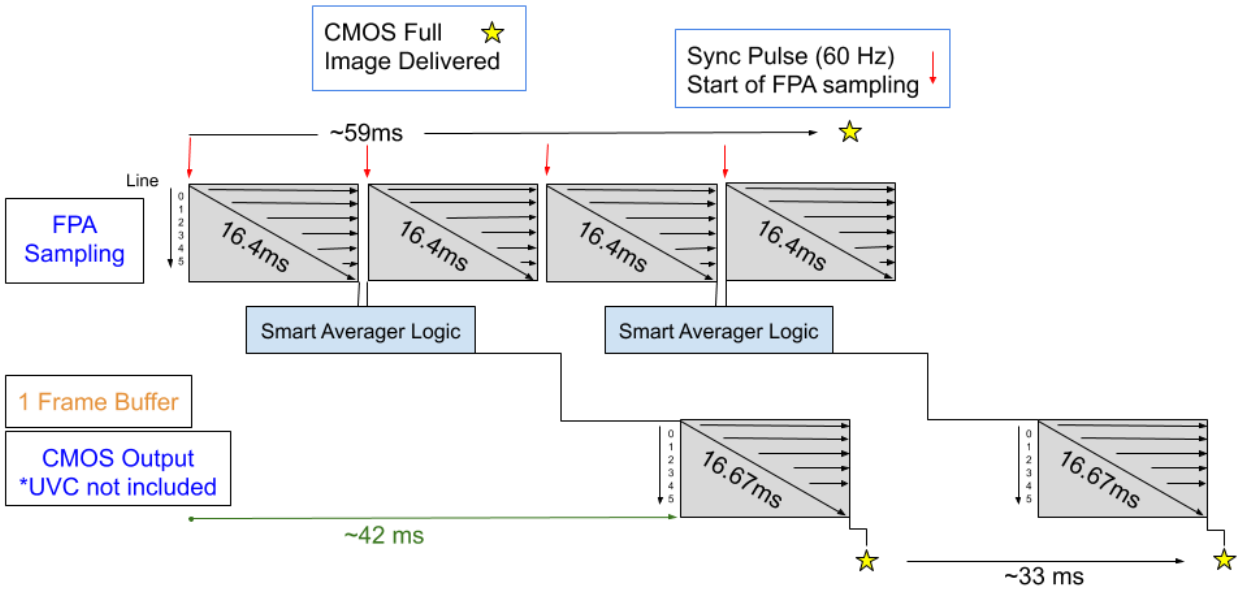


Figure 4: 16-bit Pre-AGC output averager ON - 30Hz

The information contained herein does not contain technology as defined by EAR,15 CFR772, is publicly available, and therefore not subject to EAR.

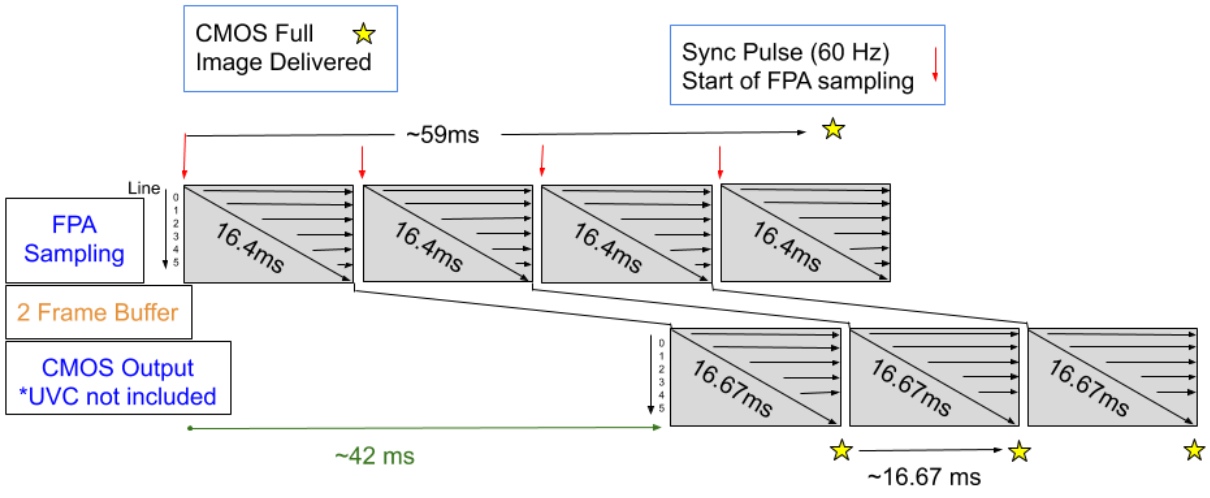


Figure 5: 8-bit post-AGC Output with averager OFF - 60Hz

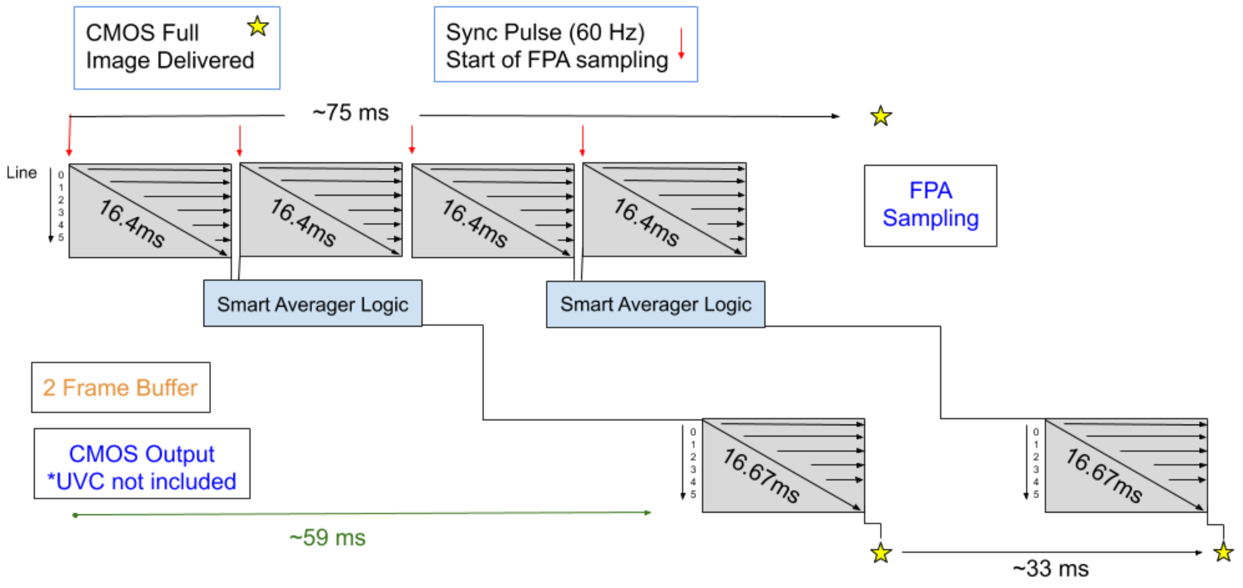


Figure 6: 8-bit Post colorized Averager ON 30Hz - USB video

The information contained herein does not contain technology as defined by EAR,15 CFR772, is publicly available, and therefore not subject to EAR.