Toshiba G2 module implementation guide

Olav Schwartz
olav.schwartz@raytrix.de

15th June 2016

raytrix

Confidential

Version	Date	Author	Notes
0.5	14.6.2016	Olav Schwartz	first draft

Contents

1 Introduction

This guide does not try to be a complete documentation of the Toshiba Galileo 2 sensor module, but a detailed index on where to find important information. Smia++ registers are written in *italic*, so they can easily be found in the Smia++ specification. All registers above 0x3000 are vendor specific.

2 Devices

All IC's on the module can be reached via 400kHz I2C.

Device	address (7-bit)	address size	data size
G2	0x10	16 bit	8 bit
Shutter	0x0C	8 bit	8 bit
Autofocus	0x0E	8 bit	8 bit
OIS	0x3E	16 bit	8 bit

Table 1: I2C configurations

2.1 Shutter

Manual control:

```
regs8[index] = 0x02; data[index++] = 0x16;
if (openClose){
  regs8[index] = 0x06; data[index++] = 0xb4;  //open
}else{
  regs8[index] = 0x06; data[index++] = 0xb1;  //close
}
```

Listing 1: shutter mode for open/close

Global start sequence:

```
regs8[index] = 0x02; data[index++] = 0x0A;
regs8[index] = 0x06; data[index++] = 0xb1;  //open
```

Listing 2: shutter mode for global start

See [4] for additional information about the shutter driver. Complete global start sequence can be found here: 5.2.

2.2 OIS firmware

The optical image stabilizer has to be programmed before first use. To program, 128 pages of 64 bytes data need to be written into the OIS driver. The first 127 pages come from a binary file, while the last page is calibration data from the sensors NVM.
Please see [5]. Full initialization routine is found at C.

3 Initialization

3.1 Power-up sequence

Right after power-up, the G2 should receive a block of initialization data. See appendix A

3.2 CSI

See[1, 17.6.6 CSI Capability Registers] for capability registers. CSI modes should be set to *CSI_signalling_mode*=2, *CSI_lane_mode*=3 and *CSI_data_format* should set the bit depth and compression. See [1, 4.7 Output Format]. *requested_link_bit_rate_mbps* must be equal or higher than the actual bitrate.

3.3 Clocks

See [2, 2.5. Clock tree configurations] and [1, 5.3 Relationship between the External Clock Frequency and the Video Timing and Output Pixel Clock Frequencies]

2.5. Clock tree configurations

Galileo-2 employs twin PLL system.

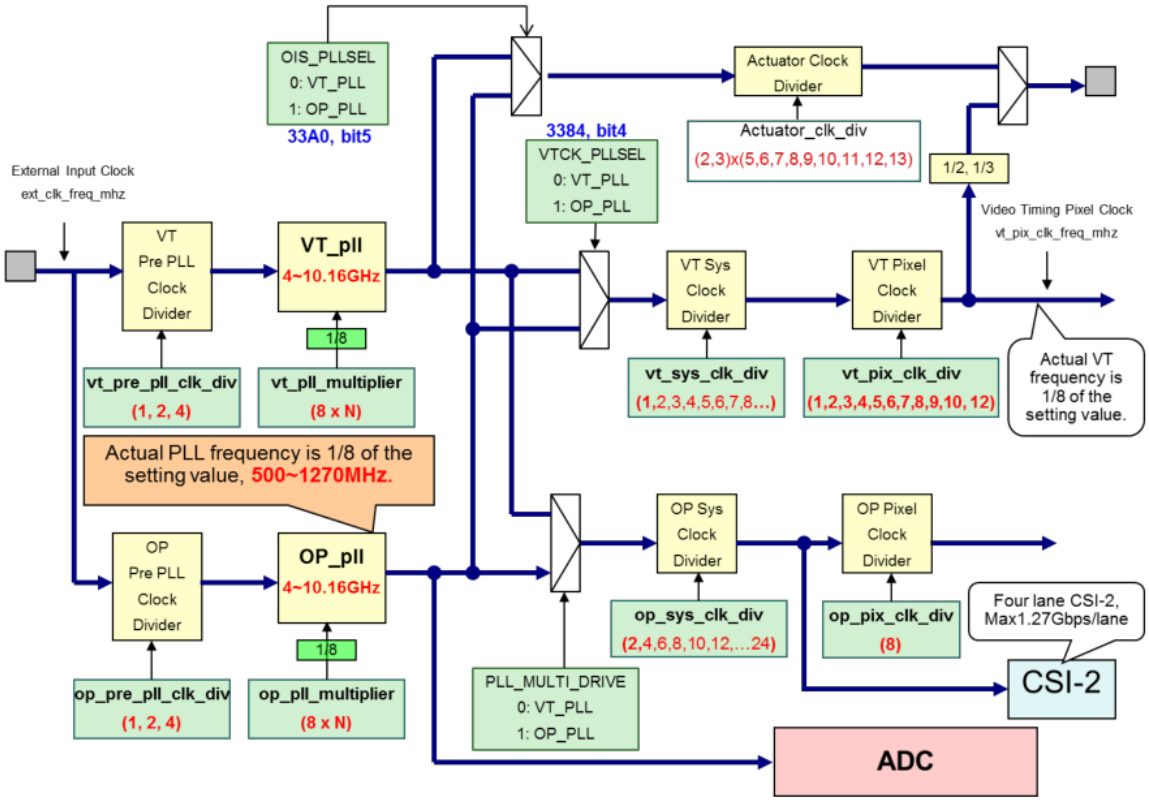

Figure 3 Clock tree

See [1, 5.3.2 Clock Set up Capability Read Only Registers] for Table 84, 85, 86, 87 for clock related capability registers. Listing 3 shows a clock configuration used by raytrix, using 19.2 MHz input clock and resulting in 531.2 MHz pixel clock.

```
vt_pix_clk_div = 6;
vt_sys_clk_div = 2;
pre_pll_clk_div = 2;
pll_multiplier = 664;
op_pix_clk_div = 10;
op_sys_clk_div = 2;
op_pre_pll_clk_div = 4;
```

Listing 3: Example Pll settings

3.4 NVM

Interfacing the NVM is completely described in [1, 10 Data Upload and Download Interface]. NVM content includes focus minx/max values

4 Video init

4.1 ROI

Image size is defined by setting ROI size, binning and scaling.

Hardware ROI size is set with (*x_addr_start, y_addr_start, x_addr_end, y_addr_end*). *x_addr_max* is 7727, *y_addr_max* is 5367. See [1, 5.2.1.1 Pixel Array Addresses] for more information. Usable pixel area is 4 pixel smaller than the maximum value.

Binning is divided in hardware- and software-binning. Hardware binning supports averaged and summed weighting (*Binning_weighting_capability, Binning_weighting*). Supported binning modes are (*binning_subtypes*) [column, row] 0x12 0x13 0x14 0x16 0x18 0x21 0x22 0x23 0x24 0x26 0x28 For additional column binning software-binning has to used. This is done by writing the software binning factor to the non-Smia++ register 0x3407.

Scaler has not been used yet.

x_output_size and *y_output_size* have to be set to the final output resolution, after binning an scaling. See [1, 4.3 Size rules] for sizing examples.

4.2 Test pattern

Various test modes are implemented in the sensor. [1, 7 Test Modes] describes all of them. Write 1 in *test_pattern_mode* to enable solid color bars.

4.3 Framerate

In rolling-shutter mode, framerate is set by calculating *frame_length_lines* and *line_length_pck* accordingly.

$1/FPS = \frac{frame\_length\_lines \cdot line\_length\_pck}{pixelclock}$

See [1, 5.2.6 Line Length and Frame Length] for value boundaries. Please keep in mind that a longer line length generates more rolling shutter effect in video mode. It's recommended to keep line length as small as possible and adjust FPS via frame length.

5 Running the G2

5.1 Rolling shutter

mode_select=1 starts camera streaming in rolling shutter mode.

5.2 Global start

Please see [1, 13 Timer Functions], appendix B.

5.3 Gain and Exposure

Please see [1, 6 Integration Time and Gain Control].

5.4 HDR

High dynamic range functionality is no part of Smia++ v. 1.00. Still, the G2 supports HDR. [3, 6. HDR] describes the feature. Config used by ratrix is HDR_mode=1 to enable HDR and Exposure_ratio=2,4,8,12. Synthesis is done on host system.

5.5 Focus

Focus is set by a 10 bit value. Focus min/max values are stored in the sensor's NVM. Please see [2, 8. NVM user area].

```
regs[index] = 0x0D80; data[index++] = (focus0_10bit >> 8) & 0x03; //   REM focus change H
regs[index] = 0x0D81; data[index++] = focus0_10bit & 0xFF;        //   REM focus change L
regs[index] = 0x0D82; data[index++] = 0x02;
regs[index] = 0x0D83; data[index++] = 0x01; //enable
xReturn = I2CWriteShortRegistersInRow(I2C_ADDRESS_G2_SENSOR, 0x0D80, data, index);
```

Listing 4: Change focus

To read data from NVM, please see 3.4

5.6 Image Header

[2, 4.2. Embedded data] shows the image header, which is sent in every image. See [1] for register definition.

References

[1] SMIA++ 1.0 Part 1: Functional specification

[2] Galileo_2_application_note_2Apr2013_forRaytrix_v092.pdf

[3] Galileo2_Sensor_spec_16Jan2012.pdf

[4] AD5830_ shutter driver.PDF

[5] OIS integration guide_rev01_23rdApr13.doc

A Initialization code

```
regs[index] = 0x130; data[index++] = 0x02;  //VANA voltage = 2.8V
regs[index] = 0x131; data[index++] = 0xCD;
regs[index] = 0x132; data[index++] = 0x01;  // VDIG voltage = 1.2V
regs[index] = 0x133; data[index++] = 0x33;
regs[index] = 0x134; data[index++] = 0x03;  //VIO voltage = 3.3V
regs[index] = 0x135; data[index++] = 0x54;
regs[index] = 0x136; data[index++] = 0x13;  //extclk = 19.2MHz
regs[index] = 0x137; data[index++] = 0x33;
regs[index] = 0x3001; data[index++] = 0x22;
regs[index] = 0x3000; data[index++] = 0x54;
regs[index] = 0x3003; data[index++] = 0x0;
regs[index] = 0x3004; data[index++] = 0x0;
regs[index] = 0x3011; data[index++] = 0x0;
regs[index] = 0x303a; data[index++] = 0x1B;
regs[index] = 0x303b; data[index++] = 0x17;
regs[index] = 0x303c; data[index++] = 0x14;
regs[index] = 0x303d; data[index++] = 0x11;
regs[index] = 0x30b5; data[index++] = 0x90;
regs[index] = 0x30fe; data[index++] = 0x00;
regs[index] = 0x3105; data[index++] = 0x30;
regs[index] = 0x3121; data[index++] = 0x20;
regs[index] = 0x312B; data[index++] = 0x80;
regs[index] = 0x312f; data[index++] = 0x30;
//black level correction registers
regs[index] = 0x3137; data[index++] = 0x11;
regs[index] = 0x313c; data[index++] = 0x10;
regs[index] = 0x313d; data[index++] = 0x02;
regs[index] = 0x3154; data[index++] = 0x01;
regs[index] = 0x3155; data[index++] = 0x07;
regs[index] = 0x3156; data[index++] = 0x11;
regs[index] = 0x3157; data[index++] = 0x25;
regs[index] = 0x3201; data[index++] = 0x00;
//defect correction
regs[index] = 0x0b05; data[index++] = 0x00;
regs[index] = 0x0b06; data[index++] = 0x01;
regs[index] = 0x0b07; data[index++] = 0x98;
regs[index] = 0x0b0a; data[index++] = 0x01;
regs[index] = 0x0b0b; data[index++] = 0x98;
regs[index] = 0x3280; data[index++] = 0x0C;
regs[index] = 0x3281; data[index++] = 0x0A;
regs[index] = 0x3282; data[index++] = 0x08;
regs[index] = 0x3283; data[index++] = 0x40;
regs[index] = 0x3284; data[index++] = 0x80;
regs[index] = 0x3307; data[index++] = 0x2C;
regs[index] = 0x3308; data[index++] = 0x20;
//begin AF calibration
regs[index] = 0x3484; data[index++] = 0x1C;
regs[index] = 0x3480; data[index++] = 0x34;

regs[index] = 0x3490; data[index++] = 0x64;
regs[index] = 0x3491; data[index++] = 0x04;
regs[index] = 0x3492; data[index++] = 0x02;
regs[index] = 0x3493; data[index++] = 0x01;
regs[index] = 0x3494; data[index++] = 0xff;
regs[index] = 0x3495; data[index++] = 0x00;
regs[index] = 0x3496; data[index++] = 0x06;
regs[index] = 0x3497; data[index++] = 0x49;
regs[index] = 0x3498; data[index++] = 0xb4;
regs[index] = 0x3499; data[index++] = 0x00;
regs[index] = 0x349A; data[index++] = 0x0C;
regs[index] = 0x349B; data[index++] = 0x00;
regs[index] = 0x349C; data[index++] = 0x00;
regs[index] = 0x349D; data[index++] = 0x00;
regs[index] = 0x349E; data[index++] = 0x00;
regs[index] = 0x349F; data[index++] = 0x00;

regs[index] = 0x3480; data[index++] = 0x35;
```

```
regs[index] = 0x3480; data[index++] = 0x04;

// set global shutter parameters
regs[index] = 0x0C02; data[index++] = 0xC1;
```

Listing 5: G2 init data

B Global start sequence

```
ToshibaG2Standby();
regs[index] = 0x0C07; data[index++] = 0x10;
regs[index] = 0x0C08; data[index++] = 0x03;
regs[index] = 0x0C09; data[index++] = 0x5A;*/
//write maximum exposure to rolling shutter mode,
//so we don't accidentally make a picture when enabling streaming.
index = 0;
regs[index] = 0x0202; data[index++] = 0xFF ;
regs[index] = 0x0203; data[index++] = 0xFF;
xReturn = I2CWriteShortRegistersInRow(I2C_ADDRESS_G2_SENSOR, 0x202, data, 2);

if (m_ShutterEnabled){
retryCounter = 0;
index = 0;
regs8[index] = 0x02; data[index++] = 0x0A;
regs8[index] = 0x06; data[index++] = 0xb1;  //open
xReturn = I2CWriteByteMultipleRegisters(I2C_ADDRESS_G2_SHUTTER, regs8, data, index);
}
xReturn |= I2CWriteShortRegister(I2C_ADDRESS_G2_SENSOR, 0x0C00, 1);
xReturn |= I2CWriteShortRegister(I2C_ADDRESS_G2_SENSOR, 0x3009, 0x20);
ToshibaG2Stream();
//start readout
xReturn |= I2CWriteShortRegister(I2C_ADDRESS_G2_SENSOR, REG_GLOBAL_RESET_CTRL2, 1);
xReturn |= I2CWriteShortRegister(I2C_ADDRESS_G2_SENSOR, 0x106, 1);
//check if readout is finished
xReturn |= I2CReadShortRegister(I2C_ADDRESS_G2_SENSOR,REG_GLOBAL_RESET_CTRL2,&ret_value);
if (xReturn != CY_U3P_SUCCESS)
{
  return ;
}
while ((ret_value != 0) && (retryCounter < 0xFFFF))
  {

    xReturn = I2CReadShortRegister(I2C_ADDRESS_G2_SENSOR,REG_GLOBAL_RESET_CTRL2,&ret_value);
    //CyU3PDebugPrint(4,"polling REG_GLOBAL_RESET_CTRL2 %d\r\n",ret_value);
    CyU3PThreadSleep(1);
    retryCounter++;
  }
  if(retryCounter == 0xFFFF)
  {
    xReturn =  CY_U3P_ERROR_DEVICE_BUSY;
  }
ToshibaG2Standby();
index = 0;
xReturn |= I2CWriteShortRegister(I2C_ADDRESS_G2_SENSOR, 0x3009, 0);
regs[index] = 0x0202; data[index++] = (m_global_c_int_time_reg >> 8) ;
regs[index] = 0x0203; data[index++] = m_global_c_int_time_reg & 0xFF;
xReturn = I2CWriteShortRegistersInRow(I2C_ADDRESS_G2_SENSOR, 0x202, data, 2);
if (m_ShutterEnabled){
xReturn |= SetShutter(1);
}
```

Listing 6: Global start sequence

C OIS firmware initialization

```
CyU3PReturnStatus_t loadOISFirmware(void ){
  CyU3PReturnStatus_t xReturn = CY_U3P_SUCCESS;
  uint16_t index = 0;
  uint8_t ret_value=0;
  uint8_t  i=0;
  uint8_t  retryCounter=0;
  uint8_t NVMCalibData[64] = {0};
  uint8_t NVMCalibPadding[11] = { 0x3F, 0x80, 0x00, 0x00, 0x3F, 0x80, 0x00 , \\
       0x00 , 0xFF , 0xFF , 0xFF};
  uint16_t ushFlashPageStart = 2015;
  uint16_t reg_page = 0x583;
  uint16_t reg_firmware_start = 0x581;
  puDataTempOIS = data; // data will be also used for OIS firmware load
  //get stuff from page 16
  xReturn = getDataFromNVM(16,40,24,puDataTempOIS);
  if (xReturn != CY_U3P_SUCCESS){return xReturn;}
  CyU3PMemCopy(NVMCalibData,puDataTempOIS,24);
  //get rest from page 17
  xReturn = getDataFromNVM(17,0,29,puDataTempOIS);
  if (xReturn != CY_U3P_SUCCESS){return xReturn;}
  CyU3PMemCopy(NVMCalibData+24,puDataTempOIS,29);
  //now add the padding
  CyU3PMemCopy(NVMCalibData+53,NVMCalibPadding,11);

  // read first 256 bytes of OIS firmware from camera flash
  CyFxFlashProgSpiTransfer(ushFlashPageStart, 256, puDataTempOIS, CyTrue);

  if (!((puDataTempOIS[0]==0x1F) && (puDataTempOIS[1] == 0x5E)))
  {
    xReturn = CY_U3P_ERROR_MEDIA_FAILURE;
    return xReturn;
  }

  //write first 64 bytes
  xReturn = I2CWriteShortRegister(I2C_ADDRESS_G2_OIS,reg_page,0);
  xReturn = I2CWriteShortRegister(I2C_ADDRESS_G2_OIS,reg_firmware_start,3);
  xReturn = I2CWriteShortRegistersInRow(I2C_ADDRESS_G2_OIS,0x584,puDataTempOIS,64);



  for (i=1;i<128;i++)
  {
    // setze den datenpointer fuer den 256 byte block um 64 weiter
    puDataTempOIS += 64;

    if((puDataTempOIS - data) ==  256)
    {
      puDataTempOIS = data;
      ushFlashPageStart++;
      CyFxFlashProgSpiTransfer(ushFlashPageStart, 256, puDataTempOIS, CyTrue);
    }

    // schreibe
    xReturn = I2CWriteShortRegister(I2C_ADDRESS_G2_OIS,reg_page,i);
    if (xReturn != CY_U3P_SUCCESS){return xReturn;}
    xReturn = I2CWriteShortRegistersInRow(I2C_ADDRESS_G2_OIS,0x584,puDataTempOIS,64);
    if (xReturn != CY_U3P_SUCCESS){return xReturn;}
    //xReturn = xReturn && I2CGetRegister(I2C_ADDRESS_G2_OIS,0x582,1,&ret_value);
    xReturn = I2CReadShortRegister(I2C_ADDRESS_G2_OIS,0x582,&ret_value);
    if (xReturn != CY_U3P_SUCCESS){return xReturn;}
    while ((ret_value != 3) && (retryCounter < 100))
    {
      xReturn = I2CReadShortRegister(I2C_ADDRESS_G2_OIS,0x582,&ret_value);
      //CyU3PDebugPrint(4,"polling page ready %d\r\n",ret_value);
    }
    if(retryCounter == 100)
    {
```

```
      return CY_U3P_ERROR_DEVICE_BUSY;
    }

    retryCounter = 0;
    ret_value = 0;


  }
  /*//write the  page (137) by hand, because we have to write to calib data from the NVM
  xReturn = I2CWriteShortRegister(I2C_ADDRESS_G2_OIS,reg_page,137);
  if (xReturn != CY_U3P_SUCCESS){return xReturn;}
  xReturn = I2CWriteShortRegistersInRow(I2C_ADDRESS_G2_OIS,0x584,NVMCalibData,64);
  if (xReturn != CY_U3P_SUCCESS){return xReturn;}
  xReturn = I2CReadShortRegister(I2C_ADDRESS_G2_OIS,0x582,&ret_value);
  if (xReturn != CY_U3P_SUCCESS){return xReturn;}
  while ((ret_value != 3) && (retryCounter < 100))
  {
    xReturn = I2CReadShortRegister(I2C_ADDRESS_G2_OIS,0x582,&ret_value);
    CyU3PDebugPrint(4,"polling page ready %d\r\n",ret_value);
  }
  if(retryCounter == 100)
  {
    return CY_U3P_ERROR_DEVICE_BUSY;
  }*/


  index = 0;

  regs[index] = 0x581; data[index++] = 0x4;              //
  //regs[index] = 0x582; data[index++] = 0x0;              //new on 24.4.2014 taken from lumia2.txt
  xReturn = I2CWriteShortMultipleRegisters(I2C_ADDRESS_G2_OIS, regs, data, index);
  xReturn = I2CReadShortRegister(I2C_ADDRESS_G2_OIS,0x524,&ret_value);
  while ((ret_value != 1) && (retryCounter < 100))
  {
    xReturn = I2CReadShortRegister(I2C_ADDRESS_G2_OIS,0x524,&ret_value);
    //CyU3PDebugPrint(4,"polling a %d\r\n",ret_value);
    retryCounter++;
  }

  if(retryCounter == 100)
  {
    return CY_U3P_ERROR_DEVICE_BUSY;
  }


/*
  ret_value = 0;
  retryCounter = 0;

  index = 0;
  regs[index] = 0x70; data[index++] = 0xC3;
  regs[index] = 0x62; data[index++] = 0xA0;
  regs[index] = 0x63; data[index++] = 0xA0;
  regs[index] = 0x68; data[index++] = 0xC0;
  regs[index] = 0x71; data[index++] = 0x00;
  regs[index] = 0x72; data[index++] = 0x00;
  regs[index] = 0x74; data[index++] = 0xC0;
  regs[index] = 0x75; data[index++] = 0x21;
  regs[index] = 0x76; data[index++] = 0x04;
  regs[index] = 0x77; data[index++] = 0x04;
  regs[index] = 0x83; data[index++] = 0x00;
  regs[index] = 0x8D; data[index++] = 0x00;
  regs[index] = 0x02; data[index++] = 0x00;
  regs[index] = 0x03; data[index++] = 0x00;
  regs[index] = 0x07; data[index++] = 0x00;
  regs[index] = 0x08; data[index++] = 0x00;
  regs[index] = 0x04; data[index++] = 0x00;
  regs[index] = 0x05; data[index++] = 0x00;
  regs[index] = 0x09; data[index++] = 0x00;
  regs[index] = 0x0A; data[index++] = 0x00;
```

```
  regs[index] = 0x115A; data[index++] = 0x00;
  regs[index] = 0x119A; data[index++] = 0x00;

  regs[index] = 0x520; data[index++] = 0x01;
  xReturn = I2CWriteShortMultipleRegisters(I2C_ADDRESS_G2_OIS, regs, data, index);
  while ((ret_value != 1) && (retryCounter < 100)){
    xReturn = I2CReadShortRegister(I2C_ADDRESS_G2_OIS,0x524,&ret_value);
    CyU3PDebugPrint(4,"polling c %d\r\n",ret_value);
    retryCounter++;
  }


  if(retryCounter == 100)
  {
    return CY_U3P_ERROR_DEVICE_BUSY;
  }*/


  //Initialization
  ret_value = 0;
  retryCounter = 0;


  index = 0;
  regs[index] = 0x523; data[index++] = 0x1;
  regs[index] = 0x525; data[index++] = 0x90;
  regs[index] = 0x525; data[index++] = 0x41;
  regs[index] = 0x523; data[index++] = 0x00;
  regs[index] = 0x525; data[index++] = 0x90;
  regs[index] = 0x525; data[index++] = 0x41;

  xReturn = I2CWriteShortMultipleRegisters(I2C_ADDRESS_G2_OIS, regs, data, index);
  return xReturn;
}
```

Listing 7: OIS firmware initialization