**ECE 437 Final Report**

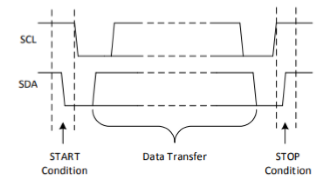
SeongJe Jung(sjung13), SeungHyun Shin(sshin14)

**Introduction**

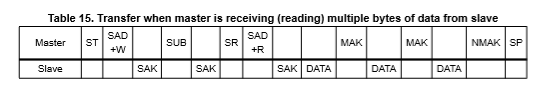
The main goal of this project is to develop the necessary software and firmware to track an object by using the CMV300 image sensor and the PMOD sensor. In order to get data from the CMV300 image sensor, we will implement SPI protocol to read and write necessary data inside the sensor. Moreover, we will also implement I2C protocol to operate the Acceleration sensor to measure the acceleration value while the system is tracking the object. In the final project, we will have 5 modules: Top module, image module, I2C module, motor module, and clock generator.

**Module description**

1. **I2C protocol module**



In order to operate I2C protocol, Start and Stop conditions should be set for this protocol. For the Start Condition, both SCL and SDA are set to high at first, and SDA is set to low while SCL stays high before both signals are set to low. For the Stop Condition, both SCL and SDA are set to low and SDA is changed to high while SCL stays low before both signals are set to high.

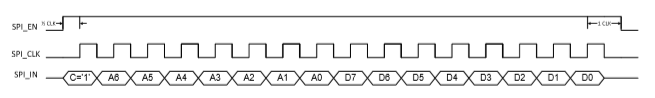


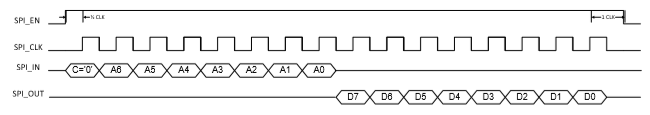
After the Start Condition, the address where data read is started from the Acceleration sensor is set. Therefore, after setting the address to 0101000 where register “OUT\_X\_L\_A” starts, data is read for 6 times so that all of the acceleration related registers up to “OUT\_Z\_H\_A” are read. While reading, SDA is set to “Hi-Z mode” to receive data from the Data Bus.



Lastly, CTRL\_REG1\_A register is set to 00110111. The 4 ODR bits are set to 0011, which is normal low-power mode with 25Hz, and set Zen, Yen, and Xen bits to high to enable the sensor to collect data of those three axes.

1. **Image module**





In order to write or read data via the SPI protocol, signals SPI\_EN, SPI\_CLK, SPI\_IN, and SPI\_OUT need to be implemented. Like I2C protocol, to write data in a particular address, the wiring command and the address to read from or write in SPI\_IN need to be set. On the other hand, to read data from the sensor, the read command is written SPI\_IN to receive data from SPI\_OUT bit by bit. With this SPI protocol, the following configurations are set.

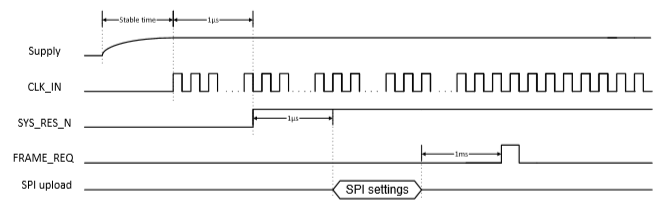
187 is written in address 83 to set the PLL\_range setting to range from 20.83MHz to 41.67MHz to use the 25MHz clock.

9 is written in address 69 to receive CLK\_OUT value from the system.

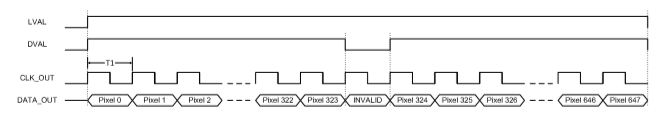
10 is written in address 42 to set the exposure time.

3 is written in address 57 wto set Output\_mode to 1 output (parallel CMOS) mode.

2 is written in address 68 to set the system to 8-bit mode.



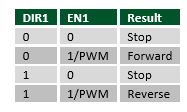
After setting the SPI configurations, FRAME\_REQ bit is sent to start collecting pixel data from the CMOS image sensor. Each pixel data contains 8-bit binary number which is written to the FIFO when the Data\_valid signal is high.



Data\_valid signal is set to low automatically after every 324 pixels and Line\_valid signal is set to low automatically after every 648 pixels. The 488 x 648 pixels make one image frame, and multiple image frames result in a video, which is collected by sending consistent Frame\_reqs to the sensor.

1. **Motor module**

The speed and direction of the motor are controlled by using the PMOD sensor.

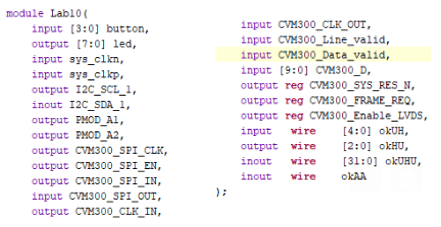


As seen in the table above, signals DIR, EN, and PWM are used to control the motor.. When DIR value is set to 0, it moves forward and when DIR value is set to 1, it moves backwards. EN and PWM signals set the speed of the motor. When EN signal is set to high, the motor moves with the speed of 1/PWM, which means that if the PWM value is increased, the speed of the motor decreases and if the PWM value is decreased, the speed of the motor increases.

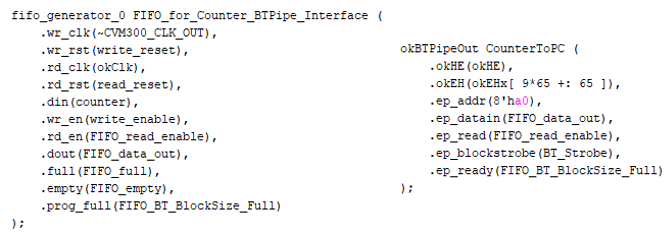
1. **Clock generator**

Inside the Clock generator, three different slow clocks are implemented to run the different FSMs. First slow clock is FSM\_Clk which is used to run the I2C FSM. The Clk\_Div is set to 10 to generate a 10 MHz clock. Second clock is Image\_Clk is used to run the SPI FSM. For this clock, the Clk\_Div is set to 4, which generates a 25MHz clock. The last clock is Motor clock with Clk\_div of 500,000, which generates a 200Hz clock.

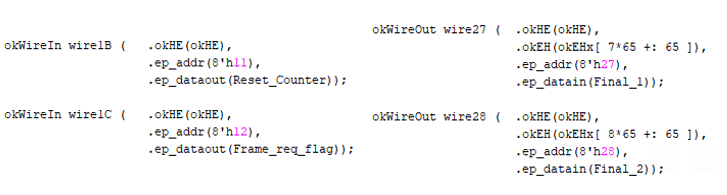
1. **Top module**

****

Inside the Top module, all the pins of the necessary sensors are connected in the FPGA board, like CVM300, PMOD, or I2C, so that data can be sent and received while operating our FSMs inside the Verilog code. When the data is ready to be sent to PC, OKWire or FIFO is used, which uses block throttled pipe to make a communication between FPGA board and PC.



For the image pixel data, FIFO which has 32-bit data\_in register is used, and FIFO will send the collected data to PC through the Block throttled pipe.



For the acceleration data, trigger values, and other input/output configuration values, OkWire is used. There are 32 available OKWire\_in registers and 32 available OKWire\_out registers and each register is 32-bit wide. The address range of Wire\_in is from 0x00 to 0x1F and the address range of Wire\_out is from 0x20 to 0x3F.

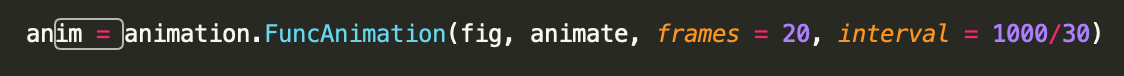
**Python description**

**Configuration():**

Using the dev.SetWireInValue(), the appropriate configurations are made from the user end to the FPGA via SPI communication.

**FuncAnimation()**:

In order to create an animation with consecutive frames obtained from the CMV300 image sensor, the function FuncAnimation from the matplotlib.animation library is implemented. By consecutively receiving the plt.imshow as its return value, the animate() method periodically sends the required reshaped array of frame data to the FuncAnimation, which displayed the received frames to the screen at 20 frames per second. The required rate of 20 frames per second is met by setting the frames and interval configurations, where frames = 20 and interval = 1000/30.



The fig parameter refers to the plt.figure in which the frames are displayed, and the animate parameter refers to the method animate() has a return value of im, which is the reshaped array of a single frame ready to be displayed. 개체이(가) 표시된 사진

자동 생성된 설명

**Animate():**

The animate() function could be seen as where all of the communications between the software and hardware are made other than the previously mentioned required configurations to the registers.

**Animate(): accelerometer**

The first technical operation is receiving the output from the accelerometer, and converting the 16 bits of data received to units of “g”. The accelerometer is triggered by simply sending the value 1 to the address 0x00 via dev.SetWireInValue, which starts the I2C communication.

텍스트이(가) 표시된 사진

자동 생성된 설명

**Fig1. First read from accelerometer**

After dev.UpdateWireOuts(), the x, y, and z data are retrieved by dev.GetWireOutValue() from registers 0x20, 0x21, and 0x22 respectively. After the unit conversion, the respective values are printed to the kernel via print(). In order to receive 100 data from the accelerometer per second, by taking advantage of the fact that the animate() function returns a frame at 20 frames per second, we simply performed the above operation 5 times throughout, which would mean that 100 data from accelerometer are received.

**Animate(): tracking with motor**

To operate the motor to the direction that an object is moving, center of mass of two consecutive frames are obtained, and the x value of the older frame is subtracted from the x value of the newer frame. Ideally, if the resulting value is positive, the motor is programmed to move the right and vice versa, but due to the noise caused by the environment, which could be the inconsistent background or possible dust and other variables between the camera and the object, it was decided to move the motor to the right when the difference is above 50 units, and left when the difference is below 50 units.

텍스트이(가) 표시된 사진

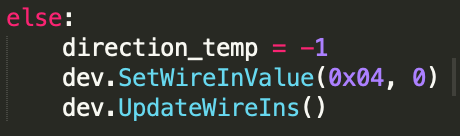
자동 생성된 설명

**Fig2. Motor operation to left direction**

텍스트이(가) 표시된 사진

자동 생성된 설명

**Fig3. Motor operation to right direction**

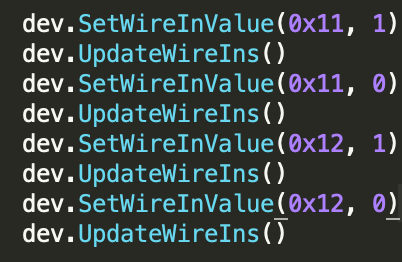


**Fig4. Motor operation with no direction**

After the direction is decided, the three variables for the successful operation of our motor are set with dev.SetWireInValue() to registers 0x01, 0x02, and 0x04, for direction, frequency, and pwm respectively. The starting condition of the motor is programmed to operate when the pwm value is a value greater than 0, therefore when there is no object motion, a value of 0 is sent to the pwm. By using the above lines of code, the operation of motor with selective directional movement is implemented.

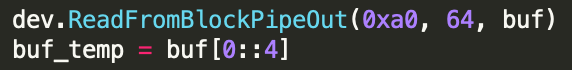
**Animate(): imager**

With the .bit file, after the correct configuration settings, to receive a frame from the CVM300 sensor, all that is needed is resetting the FIFO and sending a frame request to the image sensor.



**Fig5**. **Requesting frame from Imager**

The register 11 is where the reset value of FIFO is read, and the register 12 is where the request for frame is read by the bit file. Afer this has been complete, the array of pixel values is received by the method of dev.ReadFromBlockPipeOut.

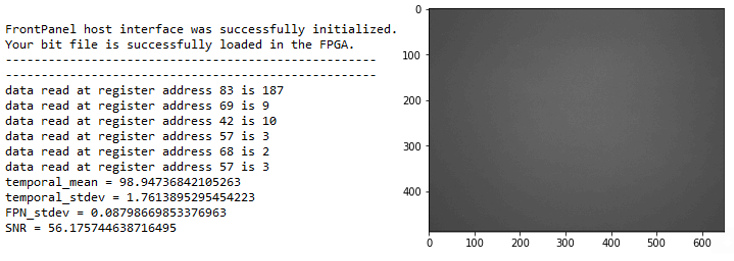


**Fig6. ReadFromBlockPipeOut**

Because the bitfile sends 32 bytes of data, with only the first 8 bytes being useful information regarding the frame, the first byte of every four bytes from the buf is stored in buf\_temp. It is this buf\_temp that is reshaped, and passed to the FuncAnimation function where the images are displayed as an animation.

**Final Lab Report Question**

1. **Spatial and temporal noise of the image sensor**



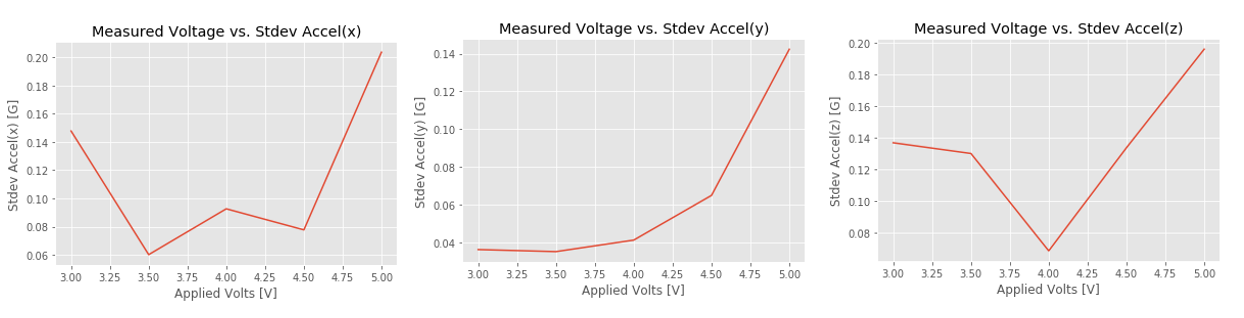
In order to get a spatial and temporal noise of our image sensor, we gather the data from the image sensor of a white wall. For the spatial noise, we calculated the average value of each pixel value from the 100 frames and find the standard deviation of the frame of average pixel values. As we can see from the data above, our spatial noise is the FPN\_stdev which is 0.0879.

For the temporal noise, we picked one pixel from the frame and find the average value of that particular pixel value of 100 frames. Then we calculated standard deviation value in order to get the temporal noise. As we can see from the data above, our temporal noise is the Temporal\_stdev which is 1.761.

1. **Signal to noise ratio of the sensor**

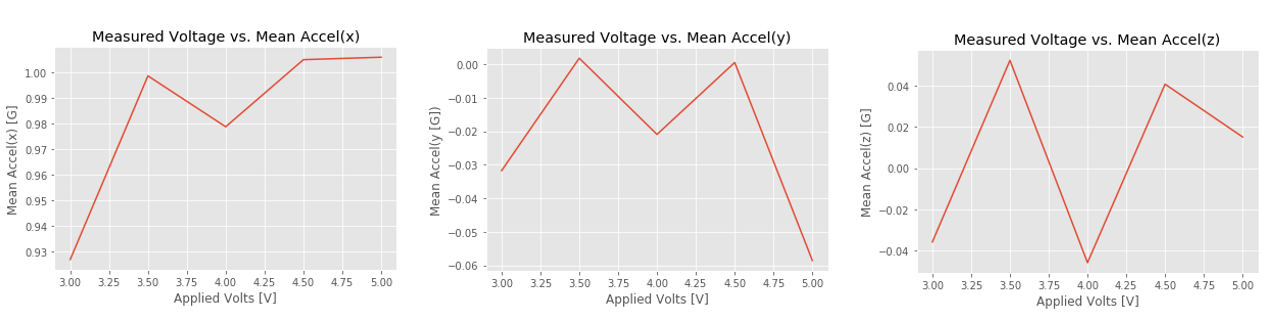
Signal to Noise Ratio, SNR, is temporal mean / temporal standard deviation value that we obtained from the pervious part. Therefore, our SNR = 98.947 / 1.761 = 56.1757. This value is also shown in the data above.

1. **Noise in the acceleration sensor**



We received the acceleration data from the sensor with the speed of 100 data per seconds. Moreover, we used five different applied voltage to the motor from 3V to 5V. In order to calculate the noise of acceleration data, we calculated the standard deviation of the data in each applied voltage. As we can see from the three graphs above, the noise increases as we increase the applied voltage on the motor. However, it seems more stable at 4V, so we decided to use this applied voltage in our final project.

1. **Acceleration of the sensor board as a function of different applied voltage on the motor**



Similar to the previous case, we received the acceleration data from the sensor with different applied voltage on the motor with the speed of 100 data per second. Then, we calculated the mean value of acceleration data in each applied voltage to get the results above. As we can see in the graphs, acceleration in X-axis stays at 1 and acceleration in Z-axis stays at almost 0 regardless of applied voltage. However, acceleration in Y-axis decreased as we moved the motor to the forward direction. As a result, we can conclude the one that is affected by the movement of the motor is the acceleration in Y-axis.

1. **Accuracy of tracking algorithms**

Our tracking algorithm works properly, and the motor is moving to the right direction. However, if we increase the motor speed by either decrease the PWM value or increase the applied voltage on the motor, the motor shakes a lot and the system become unstable. On the other hand, if we decrease the speed of the motor, the system become stable, but due to the friction of the motor, the motor does not move as much as we desire.

After we tested with several different speed of the motor, we decided to use slower motor speed to make the system stable because better stability led our system to have better accuracy.

**Appendix: codes are attached with the submission**