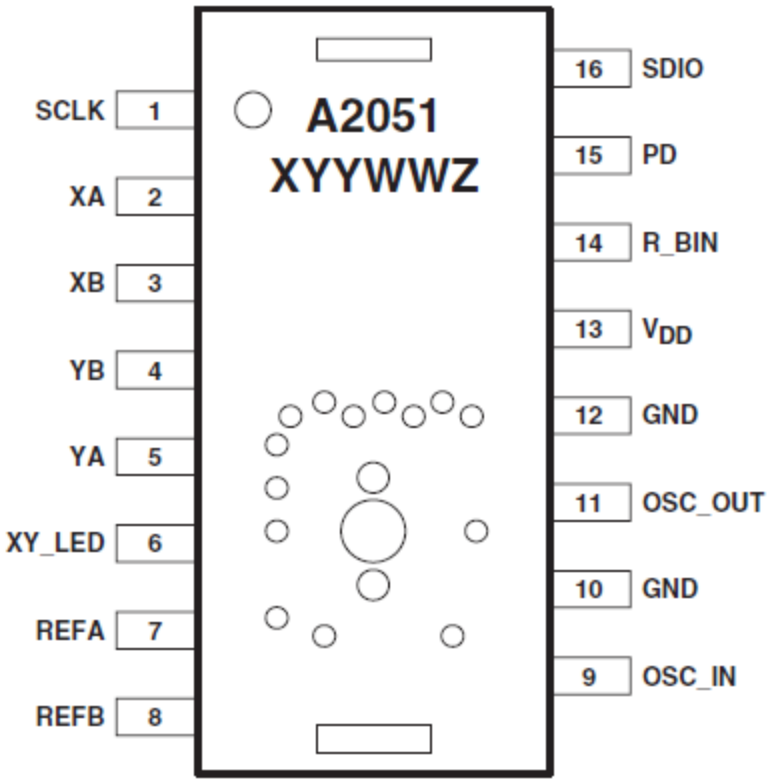
Optical Mouse Scanner
Project Design



Embedded System Design - Prof. Stephen Edwards
CSEE 4840, Spring 2013

Group Name: optical-mouse-scanner
David Calhoun (dmc2202)
Kishore Padmaraju (kp2362)
Serge Yegiazarov (sy2464)

Overview

In our project, “Optical Mouse Scanner,” we will be implementing a system in which a user can create low resolution scans of a document using an ordinary optical mouse. The mouse is operated as per usual, with the user running it over the portion of the document he/she is interested in scanning. The aggregated results of the current scan will be displayed on a computer monitor. By observing the aggregated scan on the monitor, the user will be able to note which areas of the scan are missing or erroneous, and rescan the document at those locations.

The user will be able to use the left-click and right-click functionality of the mouse to begin the scan and reset the scan, respectively. The current image being read by the optical mouse will always be displayed in a small inset box on the monitor display, next to the larger image of the aggregate scan up to that point. The position of the mouse will also be indicated on the aggregate scan. The figure below depicts the described visual output:

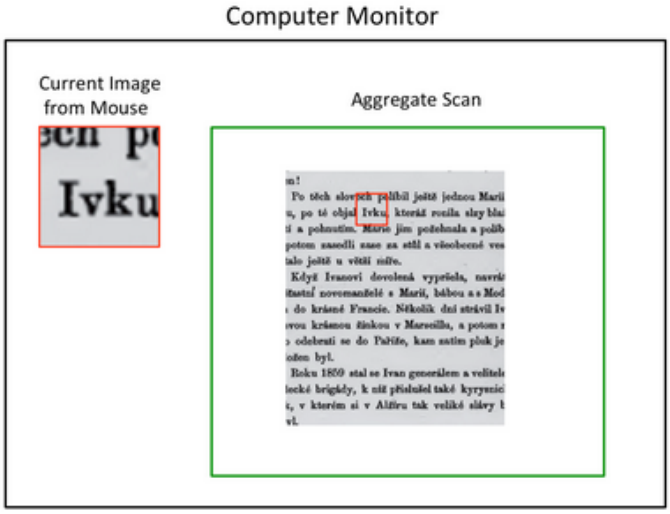


Figure 1: GUI for mouse scanner project.

A standard optical mouse transmits information about the location of the mouse in a serial, packetized format. This information can be acquired using the USB, PS/2, or direct connection to a serial pin on the processor—for the purposes of this project, the serial pin will be used. We will acquire image data from the optical mouse via the ADNS-2051 optical processor, which is one of a series of common optical processors distributed by Agilent for use in optical mice. This processor includes an 16x16 pixel CCD; the image data and XY positional data is acquired via a synchronous, half-duplex serial port on the processor. By soldering connections to the optical processor’s clock, serial I/O, and power status pins, we can communicate with and exchange data with the optical processor. The ADNS-2051 data sheet outlines communication/request protocols that we must establish to access CCD and other information. All connections to the optical processor will require soldering ribbon cables to the optical processor to establish custom GPIO (to connect to the Altera DE2).

The following image is a high-level block diagram of all the hardware components and how they interact:

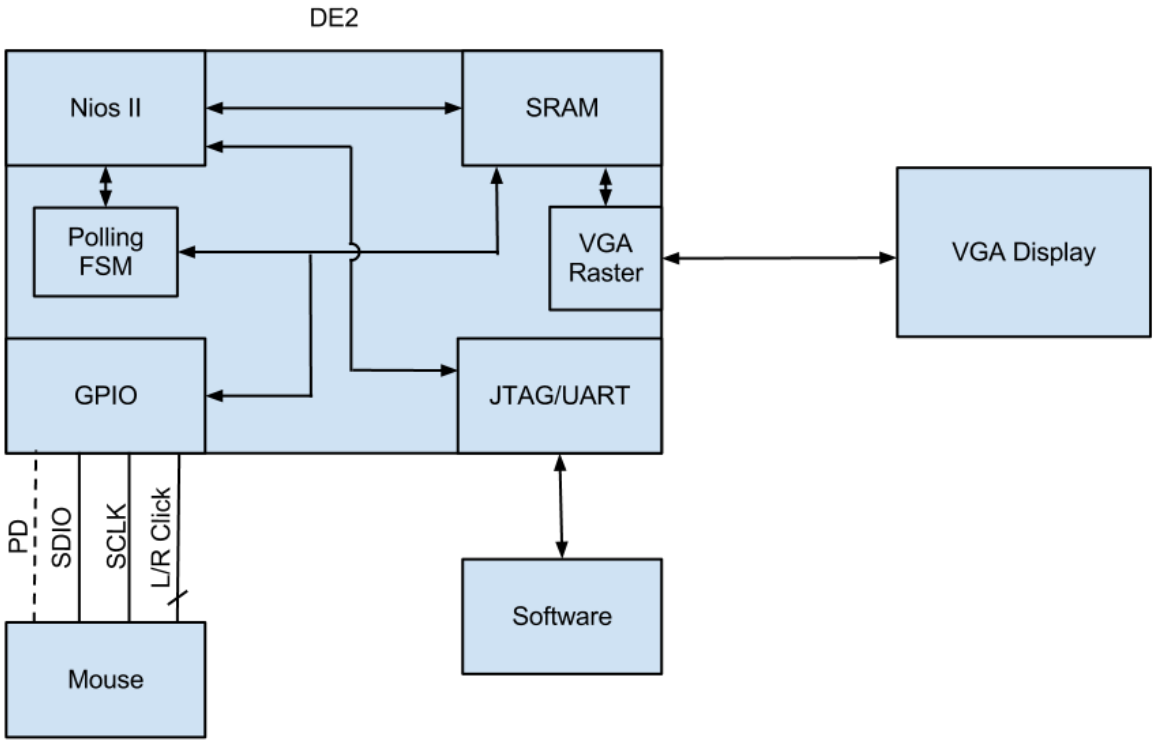


Figure 2: System architecture and hardware interconnection.

Interconnections within the DE2 block in Figure 2 indicate communication using the Avalon Bus (also known as the Avalon Switch Fabric).

Critical Path

Potential timing failure might arise when communicating between the ADNS-2051 optical processor and the DE2 board. Specifically, a polling FSM is required to communicate on the serial data line of the ADNS-2051, which will control how registers on the ADNS-2051 are accessed. The timing of how information is collected from these registers, how it is stored, and how it is used, is critical to this project design.

Specifically, the grayscale values of each pixel in a complete 16x16 scanned image sample are sequentially accessed and stored. When accessing registers that store the 6-bit grayscale values of each pixel in the 16x16 CCD image matrix, it is necessary to store all 256 pixels of the image matrix to associate with a given XY location. The given XY location itself must also be stored in a timely fashion, corresponding with the image matrix. Both of these pieces of information are stored in SRAM and accessed by software. The image and location information, stored in SRAM, is accessed via software for sorting and aggregation. This sorting and aggregation consists of displaying the current image (256 pixels) according to the current location information on the VGA display.

The exact critical path comes into play when associating the timing between polling the ADNS-2051 registers, storing/updating that information in SRAM, and accessing that information to aggregate on the VGA display. Concern rises when considering how quickly information will be collected from the mouse and updated in SRAM on the DE2, because that limits how quickly this information can be accessed and sent

to the VGA display. These timing issues can be remedied with the use of buffers.

Memory Management:

Pixel Address Map

The 16x16 CCD image pixel mapping corresponds to how data is read from the ADNS-2051, and how it physically corresponds to the captured image. The captured image is addressed from bottom right (first pixel), proceeding upwards with increasing address, eventually ending at the top left (last pixel) of the matrix shown below. Eight bits are required to represent all 256 pixels.

LAST PIXEL

FF	EF	DF	CF	BF	AF	9F	8F	7F	6F	5F	4F	3F	2F	1F	0F
FE	EE	DE	CE	BE	AE	9E	8E	7E	6E	5E	4E	3E	2E	1E	0E
FD	ED	DD	CD	BD	AD	9D	8D	7D	6D	5D	4D	3D	2D	1D	0D
FC	EC	DC	CC	BC	AC	9C	8C	7C	6C	5C	4C	3C	2C	1C	0C
FB	EB	DB	CB	BB	AB	9B	8B	7B	6B	5B	4B	3B	2B	1B	0B
FA	EA	DA	CA	BA	AA	9A	8A	7A	6A	5A	4A	3A	2A	1A	0A
F9	E9	D9	C9	B9	A9	99	89	79	69	59	49	39	29	19	09
F8	E8	D8	C8	B8	A8	98	88	78	68	58	48	38	28	18	08
F7	E7	D7	C7	B7	A7	97	87	77	67	57	47	37	27	17	07
F6	E6	D6	C6	B6	A6	96	86	76	66	56	46	36	26	16	06
F5	E5	D5	C5	B5	A5	95	85	75	65	55	45	35	25	15	05
F4	E4	D4	C4	B4	A4	94	84	74	64	54	44	34	24	14	04
F3	E3	D3	C3	B3	A3	93	83	73	63	53	43	33	23	13	03
F2	E2	D2	C2	B2	A2	92	82	72	62	52	42	32	22	12	02
F1	E1	D1	C1	B1	A1	91	81	71	61	51	41	31	21	11	01
F0	E0	D0	C0	B0	A0	90	80	70	60	50	40	30	20	10	00

FIRST PIXEL

Figure 3: 16x16 CCD image, physical pixel map and addresses.

Registers

The Data_Out_Lower register holds the values associated with the current pixel address being read. The most significant bit (MSB) of this register holds a flag that indicates whether the data is valid (meaning if it's currently being read or not) - it is high when invalid. Once a read is completed, the register is loaded up with the next pixel value and the most significant bit is set back to low, indicating that the first six bits are ready to be read once again. This cycle continues until the entire pixel map is handled:

Data_Out_Lower Address: 0x0c
Access: Read Reset Value: undefined

Bit	7	6	5	4	3	2	1	0
Field	DO_7	DO_6	DO_5	DO_4	DO_3	DO_2	DO_1	DO_0

Figure 4: Data_Out_Lower Register.

The Configuration_bits register allows us to trigger the pixel dump of the pixel array map:

Configuration_bits Address: 0x0a
Access: Read/Write Reset Value: 0x00

Bit	7	6	5	4	3	2	1	0
Field	RESET	LED_MODE	Sys Test	RES	PixDump	Reserved	Reserved	Sleep

Data Type: Bit field

USAGE: Register 0x0a allows the user to change the configuration of the sensor. Shown below are the bits, their default values, and optional values.

Figure 5: Configuration_bits register.

Because we don't want to waste buffer memory on pixel samples which match exactly previous pixel samples (if the mouse has not moved since last polling), we will need to use the motion register to first determine if any motion has occurred before reading:

Motion Address: 0x02
Access: Read Reset Value: 0x00

Bit	7	6	5	4	3	2	1	0
Field	MOT	Reserved	FAULT	OVFY	OVFX	Reserved	Reserved	RES

Data Type: Bit field

USAGE: Register 0x02 allows the user to determine if motion has occurred since the last time it was read. If so, then the user should read registers 0x03 and 0x04 to get the accumulated motion. It also tells if the motion buffers have overflowed and whether or not an LED fault occurred since the last reading. The current resolution is also shown.

Figure 6: Motion register.

The Delta_X register provides a signed representation of the amount of x-axis movement that the mouse experienced since last polling:

Delta_X Address: 0x03
Access: Read Reset Value: 0x00

Bit	7	6	5	4	3	2	1	0
Field	X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0

Data Type: Eight bit 2's complement number.

USAGE: X movement is counts since last report. Absolute value is determined by resolution. Reading clears the register.

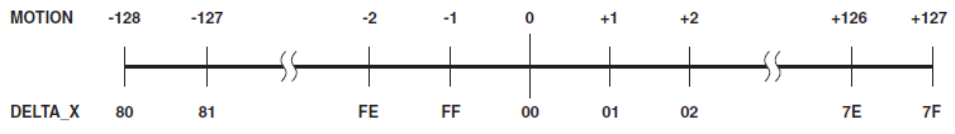


Figure 7: Delta_X Register.

The Delta_Y register provides a signed representation of the amount of y-axis movement that the mouse experienced since last polling:

Delta_Y Address: 0x04

Access: Read Reset Value: 0x00

Bit	7	6	5	4	3	2	1	0
Field	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0

Data Type: Eight bit 2's complement number.

USAGE: Y movement is counts since last report. Absolute value is determined by resolution. Reading clears the register.

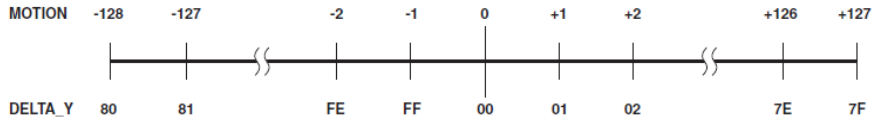


Figure 8: Delta_Y Register.

Pixel Sample Buffer Queue

In order to deal with the timing issues inherent in continual dumping of the pixels (the rest of the system may not be able to keep up), some sort of backlog or history of pixel samples will be necessary so that they may be referenced later. For this purpose, we have designed the concept of a pixel sample buffer queue. One can picture this by thinking of each pixel sample as a piece of a jigsaw puzzle, and the buffer queue as being a constantly shifting collection of these jigsaw pieces placed one atop the other. As the system continues to poll the mouse, additional jigsaw pieces are placed on top of this collection (assuming the mouse has moved). Simultaneously, jigsaw pieces are being pulled from the bottom of the collection and being placed one by one into the main puzzle, or the aggregate image of the scan. This is basically a stack.

Numerical Analysis

Because the image being pulled from the CCD is in grayscale, we will only require 6 bits per pixel for each of our pixel samples. The video frame buffer, or the aggregate image, is 128 by 128 pixels, meaning it will support 98,304 bits and 12,288 bytes. Since we are also displaying a small inlet image corresponding to the current pixel sample, we will need an additional 16 x 16 pixels = 1536 bits = 192 bytes. This comes out to a total of 12,288 + 192 = 12,480 bytes. We now need to add to this the memory required by the pixel sample buffer queue. We intend for the queue to hold at most five pixel samples - this corresponds to 192 x 5 = 960 bytes, plus one more for the 8 bits which correspond to the serial data input from the mouse peripheral. Adding this to our previous total, we get a grand total of 13,441 bytes.

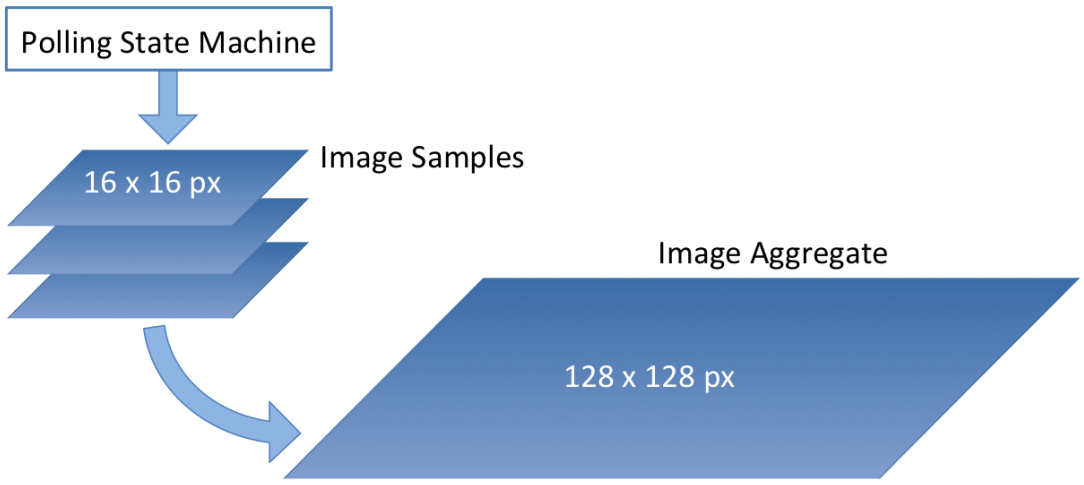


Figure 9: The polling state machine generates image samples, which are then processed into the image aggregate.

Aggregate and Inset Image Dynamics

There are essentially two groups of registers which handle all the imaging data required for display. The first group of registers handles the 64 x 64 pixel inset screen which displays the current pixel dump of whatever image the mouse is sitting on. The second group of registers handles the 128 x 128 pixel accumulation of all these pixel dumps, which is built up as the user progresses over the to-be-scanned image.

This procedure is handled with a relatively simple algorithm:

```
while (true){
        //we first fetch the front most pixel sample in the buffer queue
        pixelDump = fetchNewPixelDump();

        //we get the x coordinate corresponding to the center of the pixel sample
        pixelDumpCenterXCoordinate = getXCoordinatePixelDumpCenter(pixelDump);

        //we get the y coordinate corresponding to the center of the pixel sample
        pixelDumpCenterYCoordinate = getYCoordinatePixelDumpCenter(pixelDump);

        //we add the pixel sample to the aggregate in the location corresponding to the center of the sample
        aggregateImage.addToAggregateImage(pixelDumpCenterXCoordinate,
        pixelDumpCenterYCoordinate, pixelDump);
}
```

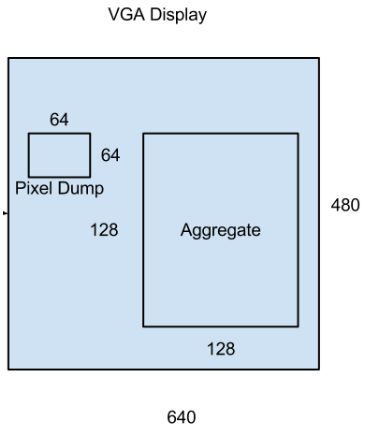


Figure 10: VGA Display—resolution for frame buffer memory allocation.

Peripherals

Optical Mouse Input Peripheral and GPIO

To connect to the optical mouse hardware peripheral shown in Figure 11, we must use the general purpose input output (GPIO) peripheral. The GPIO peripheral provides several configurable analog and digital IO pins for use with external hardware. These pins are configurable as inputs, outputs, or inouts, and can be mapped to memory. To connect the image, position, and set/reset information for capturing and displaying images to the VGA display, we required connecting several pins from the optical mouse's ADNS-2051 processor to the Altera DE2 Board via GPIO.

Five GPIO pins are required to connect to SCLK, PD, and SDIO on the ADNS-2051, and the left and right click buttons (L/R) of the mouse. These serial pins are addressed according to their physical location on a 40-pin connector on the DE2. The exact address mapping—which pins we will use—is still under consideration.

Each pin is a serial connection—SCLK and SDIO are part of a serial peripheral interface (SPI) protocol connection/communication, and PD, L, and R are enables for the polling state machine. PD is a device enable that is always set high—some optical mice can use this enable to enable low power mode. L and R will be low/high depending on if the left and right mouse buttons are pressed/not pressed (active low). The GPIO will communicate all values obtained/written to its pins via the Avalon Bus to/with other peripherals. These communications include sending PD, L, and R to the polling state machine, and writing/reading data to and from SRAM over SDIO. The typical operating frequency for the serial clock port on the ADNS-2051 is 4.5 MHz, which means that a PLL will be used to communicate a 4.5MHz clock over SCLK on the GPIO to the ADNS-2051. Figure 12 shows the interconnection of GPIO to the mouse hardware.

Left and right click controls three states of operation for the system: idle, scanning, and reset. When the left button is pressed on the mouse, the mouse will enter a scanning state, where image and position information is relayed through the GPIO to SRAM. When the right mouse button is pressed, the system resets, clearing the VGA display—this takes precedence over left click. When neither button is pressed, the system is in an idle state, where is does not poll the ADNS-2051 for new image or position data. These states are shown in Figure 13.

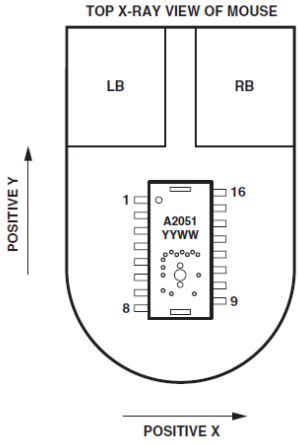


Figure 11: Mouse peripheral with embedded ADNS-2051 optical processor.

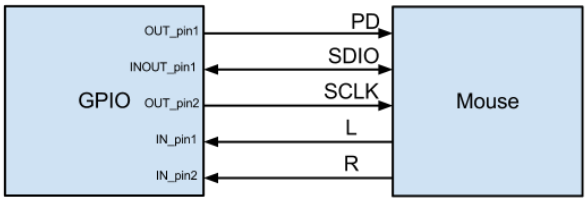


Figure 12: Mouse peripheral hardware to GPIO interconnection.

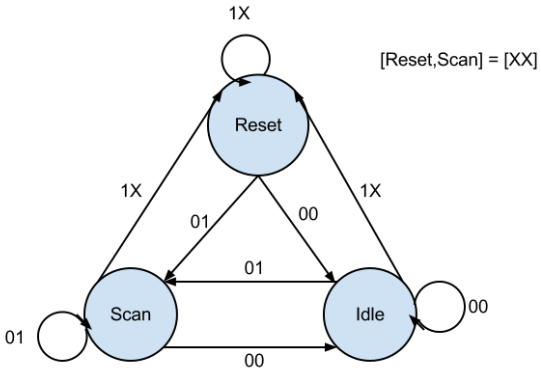


Figure 13: FSM for left and right click.

Polling State Machine

To interface with the mouse peripheral a polling state machine (PSM) is implemented in hardware on the DE2 board. The details of the PSM are given in Figure 14. As described before, the PSM communicates with the ADNS-2051 using the GPIO peripheral of the DE2. Implemented in hardware will be a serial peripheral interface (SPI) protocol for writing to and reading the registers of the ADNS-2051. The registers of

interest have already been described in the prior text. As shown in Figure 14, the main loop of the PSM continually polls the Motion register of the mouse. Within the Motion register, the MOT bit is raised high to indicate that the mouse has moved. The PSM then reads the relative X and Y movement of the mouse and transcribes the information to a new image sample stored in the SRAM. The PixDump bit in the Pixel Dump register is then set high to initiate the Pixel Dump from the ADNS-2051. As described before, the Data_Out_Lower register will continually feed the progressing pixel values for the pixel map (Figure 3). These pixel values are stored in the appropriate location of the image sample in the SRAM. Once the full pixel map has been read the PixDump bit is reset and the loop reiterates.

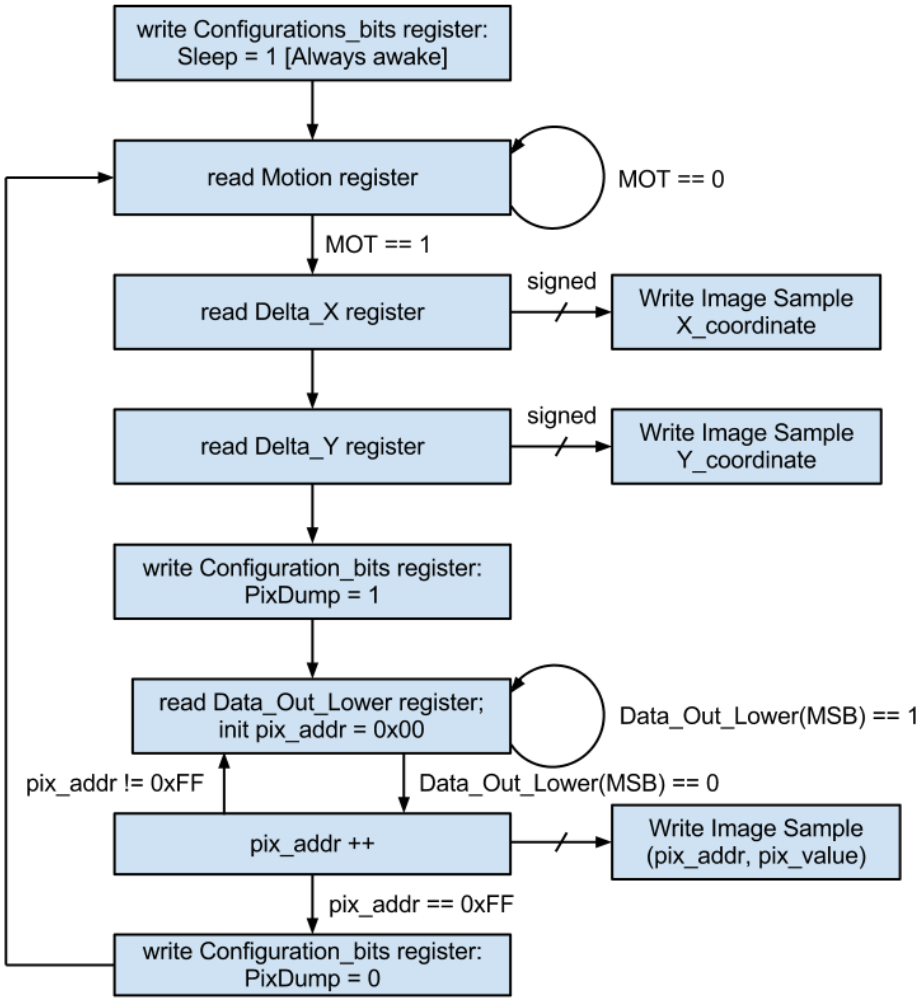


Figure 14: Algorithm for acquiring image samples from ADNS-2051

VGA Monitor Output

To display the image information on the VGA display, we require a VGA raster controller similar to the one implemented in Lab 3. The controller will update pertinent VGA signals, including the clock (~25 MHz, H_SYNC, V_SYNC, BLANK, SYNC, and the RGB level of the current VGA pixel. Once implemented in hardware (via VHDL), this peripheral will be controlled with software that updates the VGA signals. The VGA signals are stored and accessed as registers and/or counters. This peripheral communicates to the SRAM peripheral via the Avalon Bus.

Input obtained from the SRAM includes XY location and the current grayscale pixel value. Software is required to convert the grayscale pixel to 10-bit RGB (as required by the VGA hardware controller) and associate this pixel with its XY location according to the VGA signals. XY location is translated to VGA signals via counters for horizontal and vertical position. These counters, and other internal signals required for translating XY position to VGA display position are internal to the VGA controller. The following diagram outlines the basic VGA controller structure:

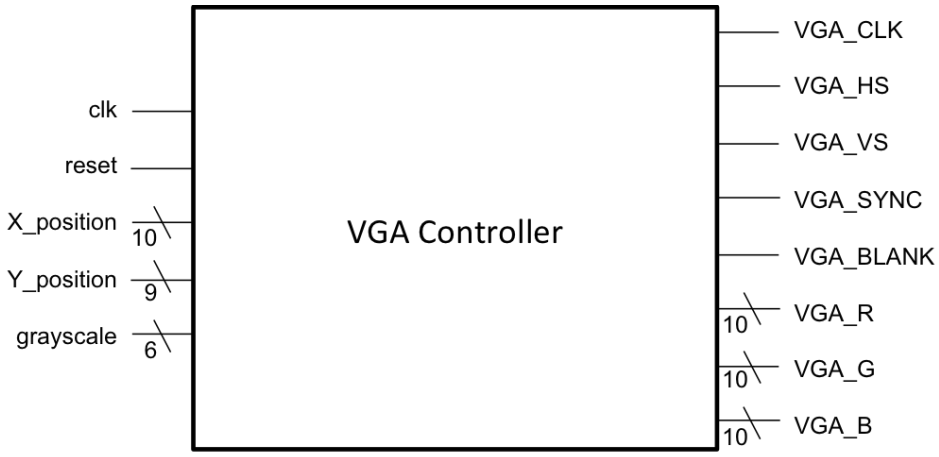


Figure 15: Block diagram of the VGA module.

References

[1] Avago ADNS-2051 Optical Mouse Sensor/Processor Data Sheet.
http://www.avagotech.com/docs/AV02-1364EN