ICPaint, A Raster Art Program

Andrew Lee
02/12/2016

Abstract
An ATMega128 was used to control a RA8875 LCD Driver Board, which in turn controlled a 800x480 pixel LCD TFT display. This hardware was used to run and display a simple raster art program with over 65,000 available colours and a small set of drawing tools. A Parallax 2-axis joystick and a KP16 16-button keypad were used as user inputs to the system, controlling a graphical 'mouse' cursor, and providing a few extra function keys. The Driver refreshes the LCD display at a frequency of 60 Hz, and the program samples the user inputs at a frequency above 50 Hz for most drawing tools, giving a smooth user interface.

1. Introduction
Simple raster (pixel based) art programs have existed since the 1980s, with the releases of MacPaint by Apple and PCPaint by Mouse Systems in 1984[1][2], and Microsoft Paint in 1985[3]. All of these programs were intended to allow the drawing of simple images on a two-dimensional grid of points, where each discrete point has its own colour value associated with it. This is known as 'Raster Graphics', and despite the near ubiquity of such systems nowadays, in the 1980s it was unclear whether such systems would be able to compete with the 'Vector Graphics' systems of the previous 20 years. Vector graphics allowed for highly accurate images, but were often limited in terms of what shapes can be drawn to a few basic geometries, such as lines, parabolas and circular arcs, and often had difficulty displaying varying colours. However, it was technology that was elegant in its simplicity, as to draw these geometries all that was needed was a phosphorus screen, an electron gun, and two controllable electric fields [4].

In contrast, raster graphics could support arbitrary geometries and colours, but were constrained by the small memories, and low resolutions possible on the hardware of the day. This made the use of raster images laughable for professional purposes, relegating it to the then small domestic market. These days, with gigabytes of memory, and displays with hundreds of pixels per inch [5], the limitations of raster displays are redundant for all but the most specialist applications. While vector graphics are still used for many things, perhaps most commonly to store the shape of each glyph in a font[6], these are still ultimately transformed, or 'rasterised', into an array of pixels before being displayed to the user.

This aim of this project was to create a simple raster art program, with a few basic functionalities, including a paint brush tool, and the ability to draw a few basic shapes, and to select the colour in which to draw. Thus the creation of an interface to enable a user to use the program was also necessary.

2. High Level Design
The User Interface (UI) for the project consists of two elements; the Graphical User Interface (GUI) displayed on the LCD screen, and the physical input devices the user manipulates. The input

devices are a Parallax 2-Axis Joystick[7], which is used to move a cursor on the screen, and a KP16 16 button keypad[8], which is used to control the speed of the cursor, to reset the program, and to 'click' on the position on screen of the cursor like a conventional computer mouse.

The hardware for the system consists of an ATMega128 Microprocessor[9], a RA8875 LCD Driver Board[10], and a 7 inch 800x480 pixel LCD screen[11]. The screen is used to display the GUI, and the image the user is creating. The image data is stored on the RA8875 LCD Driver Board (Driver), and is sent to the screen 60 times a second by the Driver. The Driver is also used to draw some basic shapes onto the image, including ellipses, rectangles and lines.

The Microprocessor itself is used to process the inputs, apply them to the GUI, and then send the resulting instructions for what to draw to the Driver. The Microprocessor also controls the communication between itself and the Driver, and stores the states of the various drawing tools.

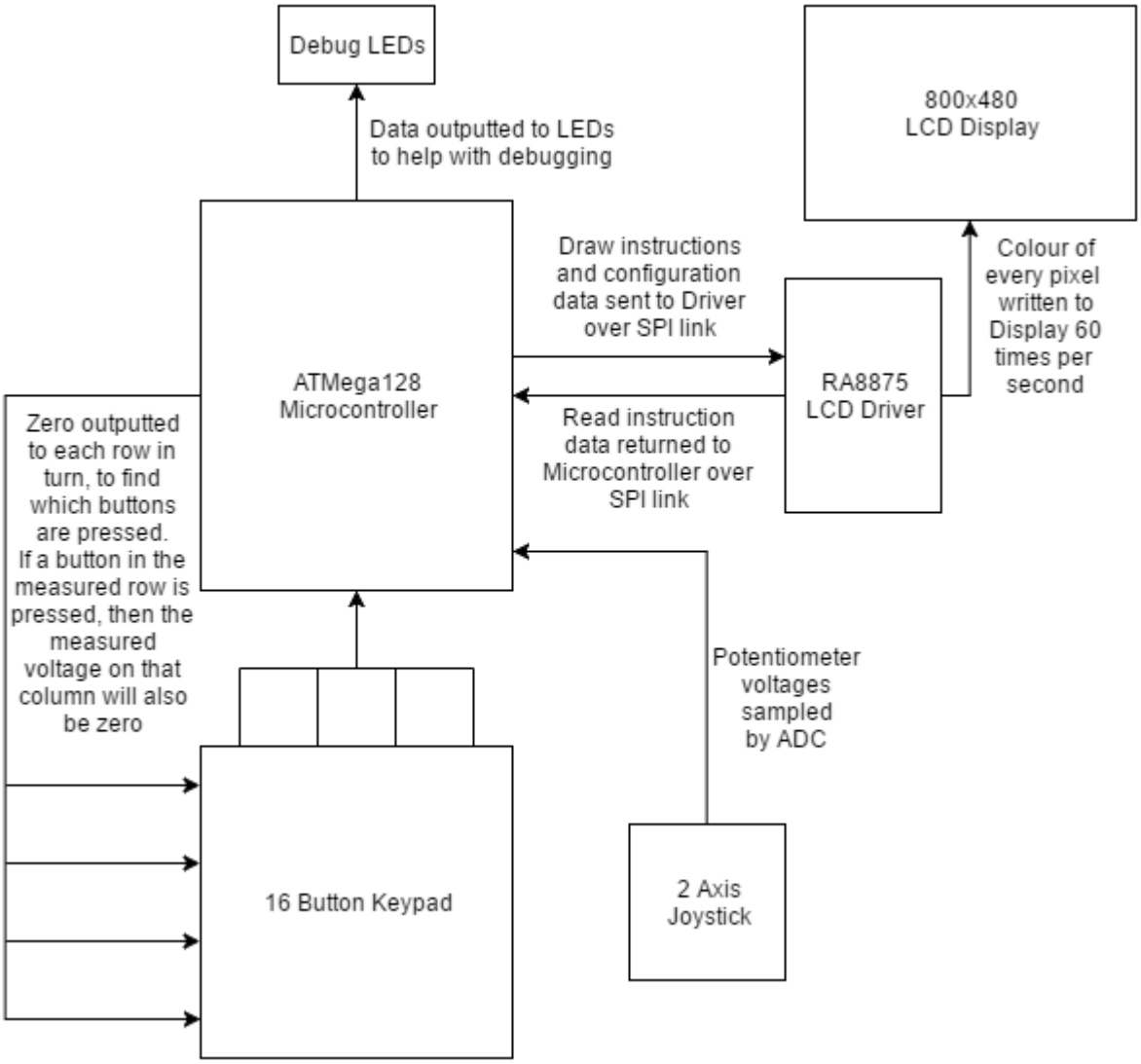


Figure 1: A simplified diagram of the flow of data between the hardware components within the project. The arrows show the direction of data flow, and the adjacent text box gives a short explanation of what data is sent along the link. There is no distinction between different communication protocols on this diagram.

A ProtoBoard [11] is also used for the connecting of the various components, and a block of 10 Light Emitting Diodes (LEDs) is used as a visual debug device. The LEDs are not intended for use in the final product, but are useful for debugging and development. A diagram showing the flow of information between the hardware devices can be found in Figure 1.

As for the software of the project, the vast majority of the code is used to interface and control the Driver. This is roughly comprised of four layers.

1. The Serial Peripheral Interface Bus (SPI) layer handles the details of passing individual bytes of information to, and reading data from the Driver
2. The Driver Data Layer builds on the SPI layer to interface with the Driver's data read/write protocol, as specified in the RA8875 datasheet ([10] page 13)
3. The Driver Instruction Layer uses the Data Layer to send instructions to the Driver, both for initialisation, and to perform most of the drawing operations onto the screen
4. The Higher Level Methods Layer uses the methods in the Driver Instruction Layer to draw shapes according to the user's inputs, draw the user interface, reset the screen, and most of the other top level features of the project

In addition to the Driver control code, there is also a method to read the state of the 16-button keypad, and a method to update the cursor position based on the joystick state.

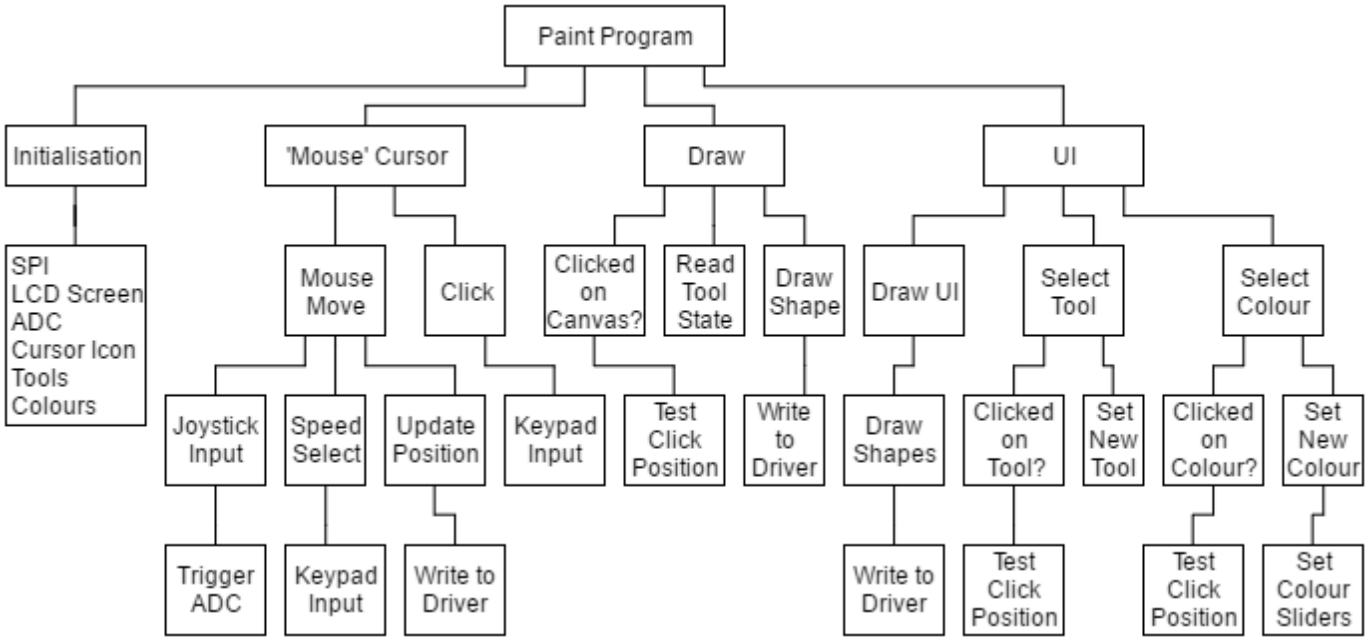


Figure 2: A top down modular diagram showing the high level structure of the code. If a box has multiple boxes connected beneath it, then these lower level elements are performed from left to right. The large box beneath 'Initialisation' lists the components of the program that need to be initialised.

The overall purpose of the software is to take the inputs from the joystick and button pad, and treat it like a conventional computer mouse. The mouse can click on a 'canvas' region of the screen to draw with the selected tool, or can click on the UI to change the selected tool, or the drawing colour. A top down modular diagram of the broad structure of the code can be found in Figure 2, and a complete hardware diagram can be found in Appendix C.

3. Software and Hardware Design

In higher-level languages, there is the concept of a function, or method, that returns some value when called according to the arguments passed to it. In assembler, this does not translate well, as there is no concept of returning with a value. Instead, some subroutines, for example ThreeBytesFromColour from GraphicalMethods.asm, have comments specifying some registers as arguments, whose values must be set to the desired value before calling, and some registers as return registers, whose values will be set by the subroutine. Every other register that is modified by the subroutine is pushed to the stack at the start of the method, and popped back again at the end, to help minimise unexpected and unwanted side effects of calling a subroutine. By doing this, subroutines in the project behave like methods in a higher-level language, and so are referred to as such in this report.

The Microprocessor is configured to communicate with the Driver using the Serial Peripheral Interface Bus (SPI), with the Microprocessor as the 'master', outputting the clock signal for the signal, and the Driver the slave, outputting data only when instructed to by the master. SPI uses four separate wires to transmit data:

- The Clock, which tells the slave when to transmit a bit of data
- The Master Out Slave In (MOSI), along which the master sends data to the slave
- The Master In Slave Out (MISO), along which the slave sends data to the master
- The Chip Select, which tells the slave whether it should act on the state of the clock, allowing the master to have multiple slaves, so long as only one is selected at a time

This is a 'full duplex' interface, meaning the master and slave both simultaneously transmit data on the rising edge of the clock. Therefore, by outputting 8 consecutive pulses on the clock, all 8 bits in the SPI data registers are transmitted from one device to the other, effectively swapping the contents of the SPI data registers on the Microprocessor and the Driver.

This interface is constructed by using the alternate function of Port B, where pins 0 to 3 are the Chip Select, Clock, MISO and MOSI pins respectively. As the Microprocessor is acting as the master in the system, the Chip Select pin is unused. The remaining 3 pins are connected to their

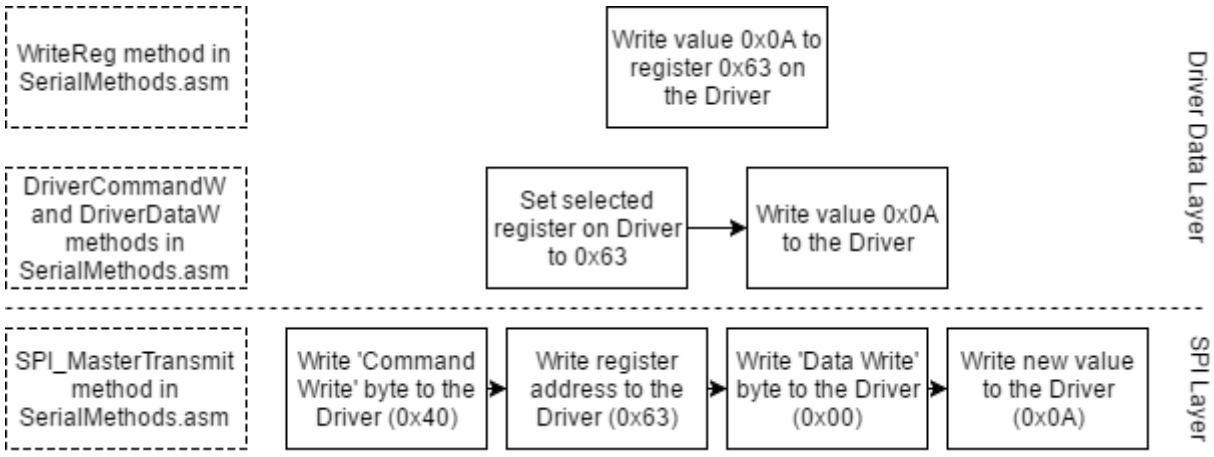


Figure 3: This diagram shows how four bytes of SPI data transfer are necessary to set the value of a single register on the Driver. On each consecutive layer down, the method used in the line above is broken down into its component methods. The dashed boxes on the left list the methods used on each layer.

corresponding pins on the Driver via the ProtoBoard. The Chip Select pin on the LCD Driver is connected to Port C on the Microprocessor, to allow the Microprocessor to control whether the Driver acts on data on the Clock and MOSI pins.

The Microprocessor then uses this interface to send pairs of bytes to the Driver. The first byte is a 'command byte', and tells the Driver what it should do with the second byte, the 'data byte'. Only the top two bits of the command byte are actually used, with the other six only serving to pad the information out to a full byte, so the command bits are cycled through to the correct positions in the SPI Data byte on the LCD Driver. The commands can either be to set the 'selected' register on the Driver, to read from, or write to the selected register on the Driver, or to read the Driver's Status Register. Thus to set the value of a specific register on the Driver, four bytes of data need to be sent over the SPI link, as shown in Figure 3. The 'WriteReg' method in the SerialMethods.asm file will do this using register 17 and 18 on the Microprocessor as arguments, allowing methods in the Driver Instruction Layer to call it instead, abstracting away a layer of complexity.

Most of the graphical operations performed by the program use this as their basis – the Driver has built in functions to draw basic shapes, and these are accessed by writing to specific registers on the Driver with the data to describe the shape. An example for drawing a rectangle is shown in Figure 4. This same approach is used to utilise the circle, ellipse, triangle and line drawing functions on the Driver. These functions all draw a shape in the Driver's 'Foreground Colour', a 16 bit value stored in a pair of registers representing a RGB colour (see Appendix A for more details). In order to draw a shape in the desired colour, these two registers must first be set to the correct value, again using the SPI link described above.

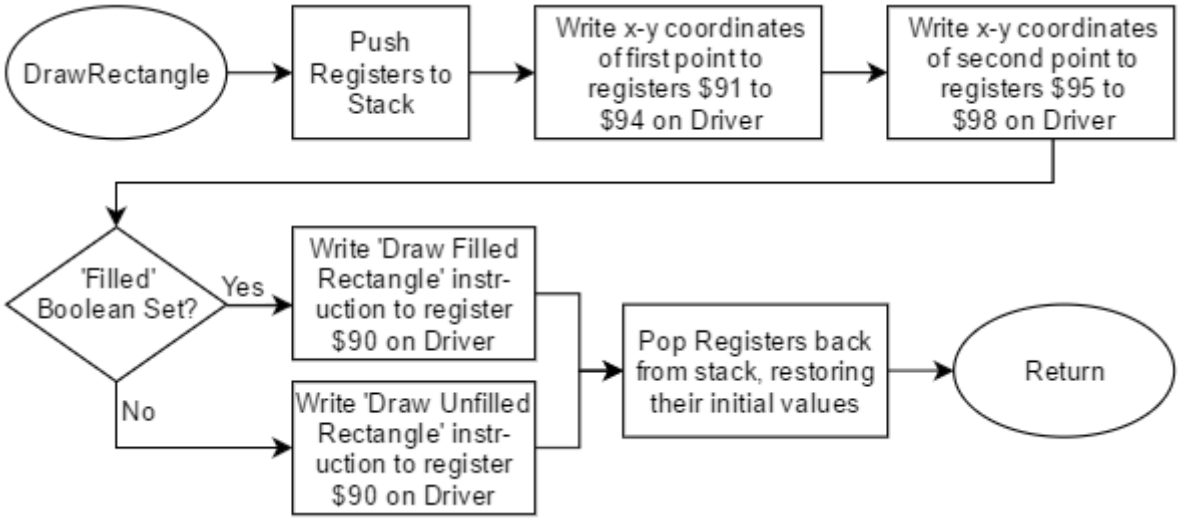


Figure 4: A flowchart showing how the DrawRectangle method from GraphicalMethods.asm functions. This demonstrates the interface between the Microprocessor and the Driver for using the drawing functions on the Driver.

The other method used to set a pixel's colour on the screen is to directly write the colour data to the memory address of the pixel, and update the colour data with a new 16-bit colour value. This approach is used in the PaintPixel method, as shown in Figure 5. A similar process is also used in the CursorShapeMouse method to write the mouse cursor icon to the Driver, though this writes

to a different block of memory, and works with a much reduced 2-bit colour space, as opposed to the 16-bit colour of the display itself ([10] page 101).

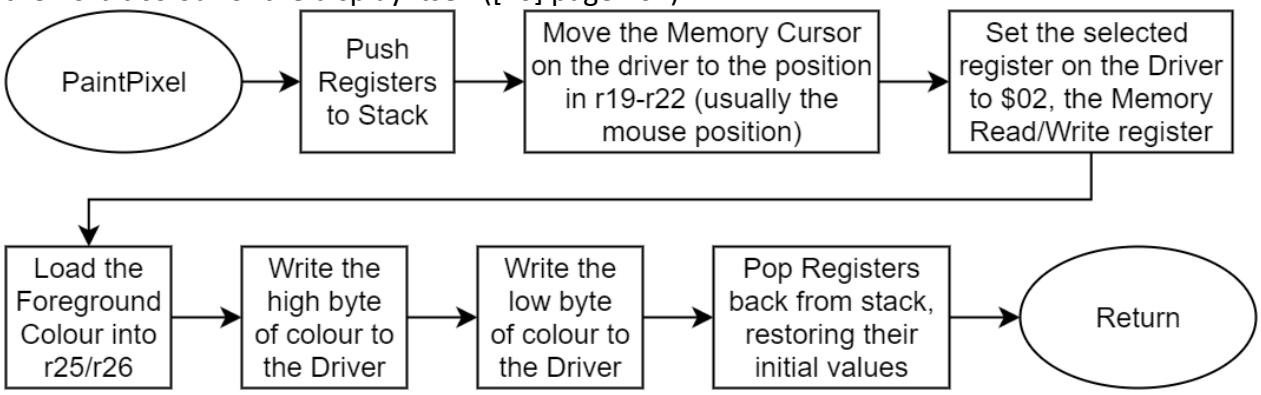


Figure 5: A flowchart showing how the PaintPixel method from GraphicalMethods.asm functions. The important feature here is that two bytes of data are consecutively written to the Driver without changing the target register between them. This is because the Driver automatically writes the data from register $02 to the target memory address set by the 'Memory Cursor' registers, $46 through $4D. Register $02 should not be thought of as a register, but as a gateway to the 768KB of RAM on the Driver.

These basic graphical objects are then used to create the entire Graphical User Interface (GUI), and are also what is drawn when the user clicks on the canvas with one of the tools. In addition to being drawn with explicitly set coordinates, as in the colour palette squares and tool icons, they can also be used to create more complex objects like the colour gradients used in the custom colour slider bars, as shown in Figure 6.

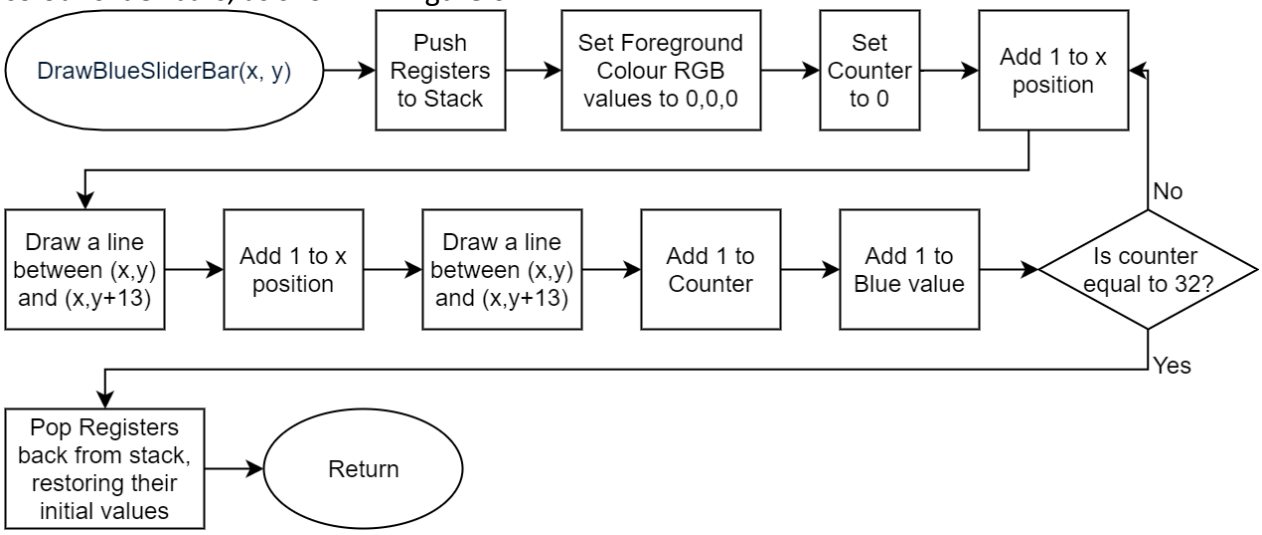


Figure 6: A flowchart showing how the blue colour slider bar is drawn on the screen, with its top left corner position specified by arguments x and y. The slider bar is 64 pixels wide for all three colours, even though there are only 32 possible values for the blue component of colour. Therefore, the code draws two pixels of each colour, by drawing two adjacent lines per loop of the code. This results in a smooth gradient being drawn from pure black up to bright blue.

The program takes advantage of the 'Active Window' on the Driver for some tools, to prevent the GUI from being painted over. This is a feature that defines a rectangle that the current drawing operations take place within, and draw instructions outside the Active Window are ignored. For example, if the Driver is instructed to draw a triangle where part of it is inside the Active Window, and part of it outside, then only the part of the triangle within the Active Window will be drawn.

Therefore, by setting the Active Window to the canvas when the user is painting, we can prevent the GUI being overwritten by the user.

The other half of the User Interface (UI) is the physical input devices the user manipulates to interact with the system. There are two separate input devices – a two-axis joystick, and a 16-button keypad. The joystick contains a pair of 10 kΩ potentiometers, one attached to the 'vertical' axis, and one to the 'horizontal' axis. This creates a two-dimensional x-y space of possible joystick positions, and so potentiometer voltage outputs. By connecting these voltages to the Analogue to Digital Converter (ADC) on the Microprocessor, we can get a 10-bit number representing the x position and a 10 bit number representing the y position of the joystick, and update the position of the cursor based on these two values. This is done by scaling the joystick values according to a mouse speed setting (in order to allow fine control for detailed shapes), then adding it to a scaled up mouse position. As shown in Figure 7, this scaling up allows for much smoother mouse movements, as it effectively allows the mouse to have fractional pixel positions, as opposed to being rigidly bound to the screen's pixel grid.

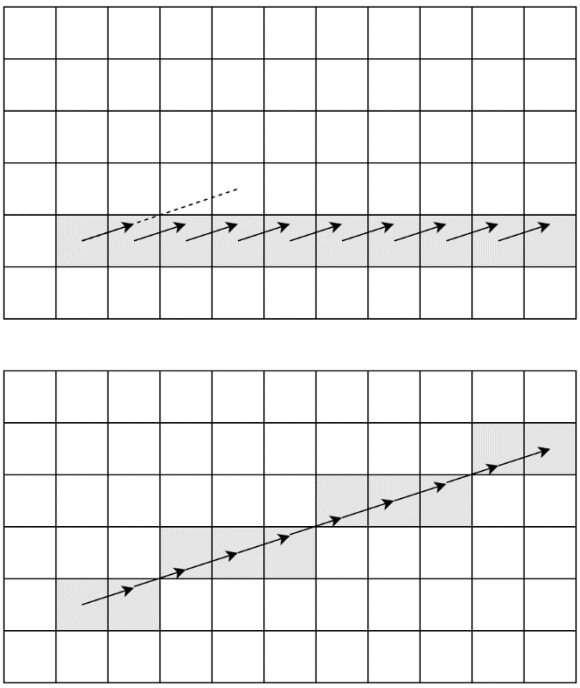


Figure 7: The top grid show the result of storing a cursor position to the same precision as the display surface. Shaded squares represent the pixels the mouse passes through, and white squares the untouched pixels. The arrow shows a scaled the vector offset from the input joystick. As shown, if the cursor is only stored to the precision of the pixels, then for a given input vector it can only travel in straight lines towards one of the 8 surrounding pixels, drastically reducing the usefulness of the 2 axis joystick, as only 8 directions are possible. In contrast, the bottom grid shows the same scenario, but where the mouse position is stored to a much higher precision than the display pixels. This allows the cursor to remember its position within a pixel, and so sporadic deviations from the 8 cardinal directions are possible, resulting in a much smoother, more flexible mouse.

In order to give the user the ability to 'click', alter the mouse movement speed, and reset the display, a 16-button keypad is used, comprising of four columns and four rows. When a button is pressed, it electrically connects its row and column. Therefore, to determine whether the '3'

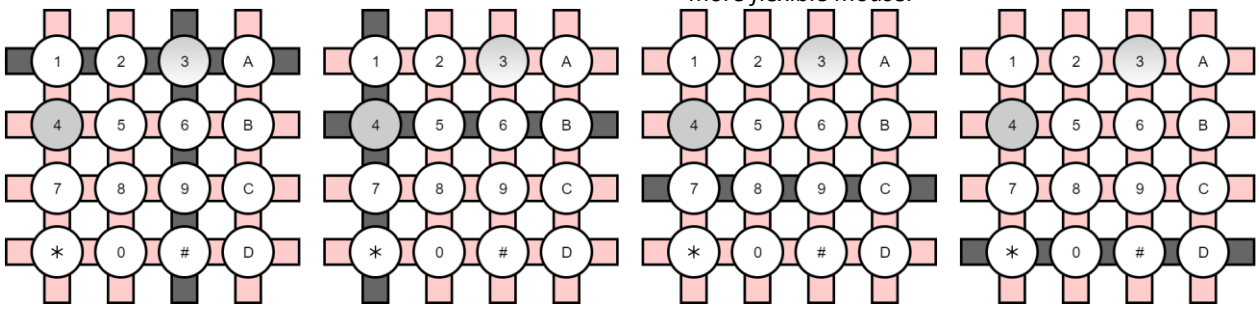


Figure 8: The pink indicates rows and columns where the pull up resistors pull the state high, and the dark grey indicates rows and columns that are electrically connected to the low output from the microprocessor. In this diagram, the states of the rows and columns are shown for all four measurements, from left to right, for the case where the '3' and '4' buttons are pressed. The first measurement tells us that button 3 must be pressed, as the third column is low. There is no other way of pulling only that column low from the first row being driven low.

button is pressed, the program drives the first row low, and connects every other row and column to a pull up resistor. This means that every other row and column will drift high unless they are connected to something low. If the '3' button is pressed, this connects the third column to the first row, pulling it down and overriding the pull up resistor. As a result, by reading the state of the four columns, we can determine if the first button is pressed, as shown in Figure 8. This process is repeated for all four rows to find which of the 16 buttons are pressed.

4. Results and Performance

Overall, the project is feature complete, as every core feature from the project plan is in the final version (see Appendix B). The one difference is that a 16-button keypad is used, rather than the originally specified QWERTY keyboard, but this does not change the functionality of the final product. In addition to the planned features, several initially unplanned UI improvements have been implemented, such selection indicators for colours and tools, and a full scale brush size indicator, along with the ability to specify the RGB values for three Custom Colours.

The single most time-consuming element of the project by far was getting the underlying SPI link and the 'Driver Data Layer' between the Driver and the Microprocessor working to the point that the Driver would turn on the LCD screen. This was partly due to a defective Microprocessor port, and partly due to loose wires making the system very unstable in its early stages.

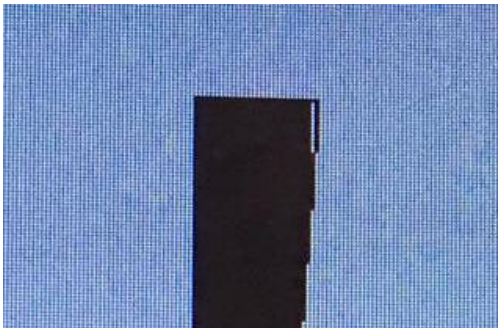


Figure 9: This photograph shows a glitch in the Triangle drawing function on the LCD Driver. The bottom of the triangle has been cropped from the photo for the sake of space efficiency.

While the use of the geometric shape functions on the Driver has saved a lot of time, as it has reduced the amount of calculations that need to take place on the Microprocessor, and reduced the lines of code needed to draw basic shapes, unfortunately it has also left the project exposed to bugs in the Driver itself. One example of this is that when drawing a filled triangle, the final column within the triangle is left unfilled, as shown in Figure 9. While this could be fixed by re-implementing the draw triangle method on the Microprocessor either by using the line drawing function or directly setting the colour of each pixel, this would have substantially worse performance than the current system in terms of speed. This is because the SPI interface has a clock speed of approximately 0.5MHz (2µs per SPI clock cycle), meaning that sending a single byte of data to a register on the Driver takes at least 64µs, as shown in Figure 10. A line requires nine bytes of data to be sent to the Driver – eight for the start and end coordinates and a ninth to initiate the drawing – meaning at least 0.57ms would be taken just to send the data for a single line in the triangle. Therefore, drawing a moderate sized triangle 100 pixels wide would take at least 0.05 seconds, without taking any consideration to the calculations required to draw it. This is not a long period of time, but is certainly long enough to be noticed by the user.

This is not just hypothetical either. On the UI, there are colour slider bars for the custom colours which have a smooth gradient from black up to pure red, green and blue. There are no supported functions on the Driver for gradients, so these are constructed line by line, slightly increasing the intensity of the colour between drawing each line. This takes a noticeable period of time, and the human eye can just about catch the 'sweep' as the lines are quickly drawn after each other. As a result, care is made to ensure the UI does not need to be redrawn very often. The only tool that requires a complete redraw of the UI is the Ellipse tool, as it does not support the Active Window on the Driver. This means that drawing an ellipse can draw on top of the UI, so the entire UI is redrawn every time the user draws an ellipse, resulting in a noticeable flickering and decrease in speed if the user draws many ellipses consecutively.

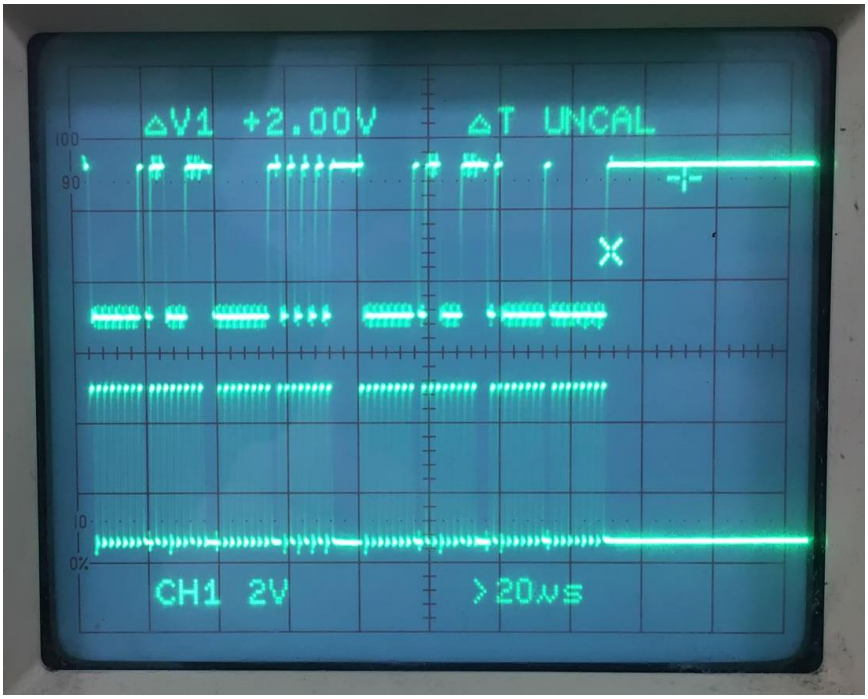


Figure 10: This photograph shows an oscilloscope where channel 1 input, at the bottom, is attached to the SPI Clock pin, and the channel 2 input, at the top, is attached to the MOSI pin. In this image, the microcontroller is writing the value $AA to register $63, then immediately reading the value of register $63 back, as part of the SerialTestLoop method in SerialMethods.asm. The clock signal is broken up into 8 distinct blocks of 8 rising edges, of which the first 4 blocks are the write operation, and the last four are the read operation. As each square across on the screen represents approximately 20μs, it can be seen that the time taken to write one byte to a register on the Driver is approximately 64μs.

Another issue is that when using the ellipse function on the Driver, the drawn ellipse can 'loop' back around the screen, as shown in Figure 11, and so parts of the screen can be painted that were not intended if the ellipse is large enough. This is presumably because the Driver is not checking whether an overflow has occurred between the defined centre of the ellipse and the point it is currently painting. The obvious solution to this would be to set the Driver's Active Window to be the rectangle bounding the ellipse, with extra bounds preventing it going off screen, as shown in Figure 12. Unfortunately, the Active Window is not supported for the ellipse drawing operation, as per the RA8875 datasheet ([10] page 98). However, it is unclear from the documentation whether the elliptical 'curve' shape supports the Active Window. If it does, then the current ellipse drawing method in the program could be modified to draw four elliptic curves instead of one whole ellipse. This would have a small performance penalty, as the Driver would need to be sent four draw instructions instead of one, but would also mean the User Interface would not have to be redrawn after every ellipse, dramatically reducing the work that must be done for an ellipse drawn on the canvas.

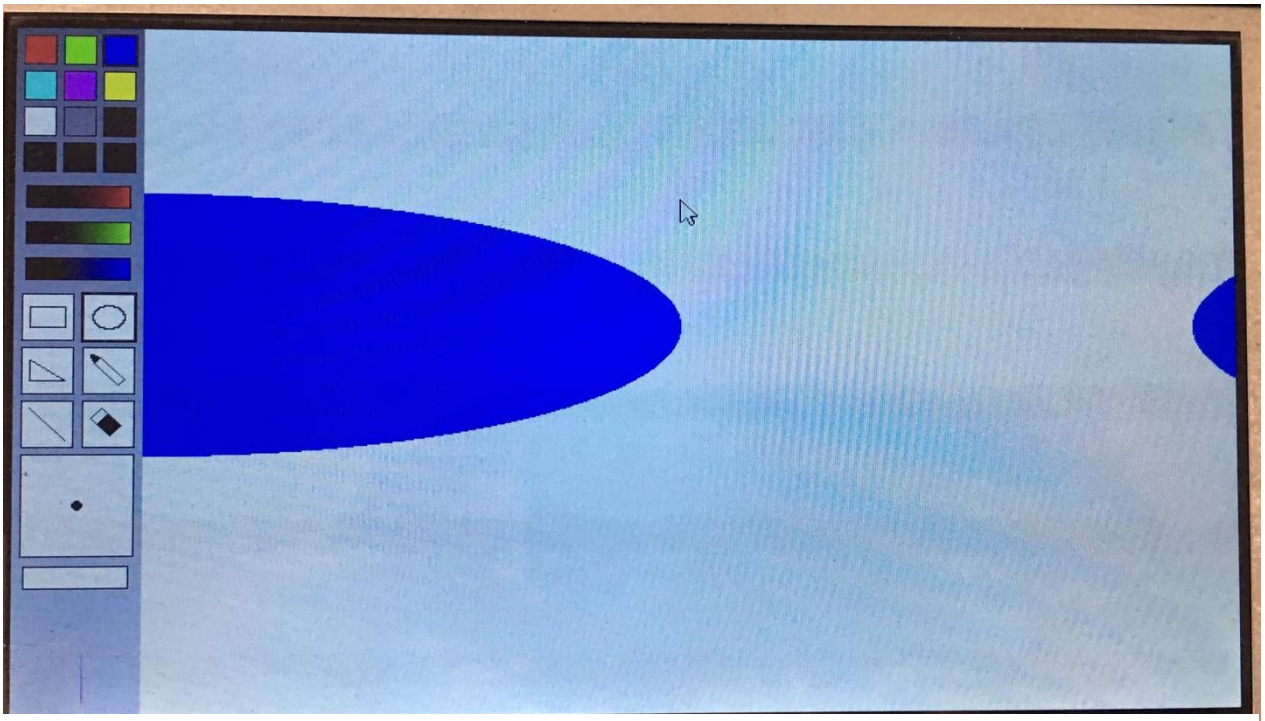


Figure 11: This photograph shows the result of drawing an ellipse with a large semi-major axis. The left-hand side of the ellipse, which goes off the x=0 side of the screen, appears to have been drawn, overflowed from 0 up to 1023, and then continued down until it passed the x=799 mark, and began displaying on screen again. The UI does not appear affected, as it is completely redrawn after every time the user draws an ellipse. A similar effect can happen with an ellipse with a large vertical axis.

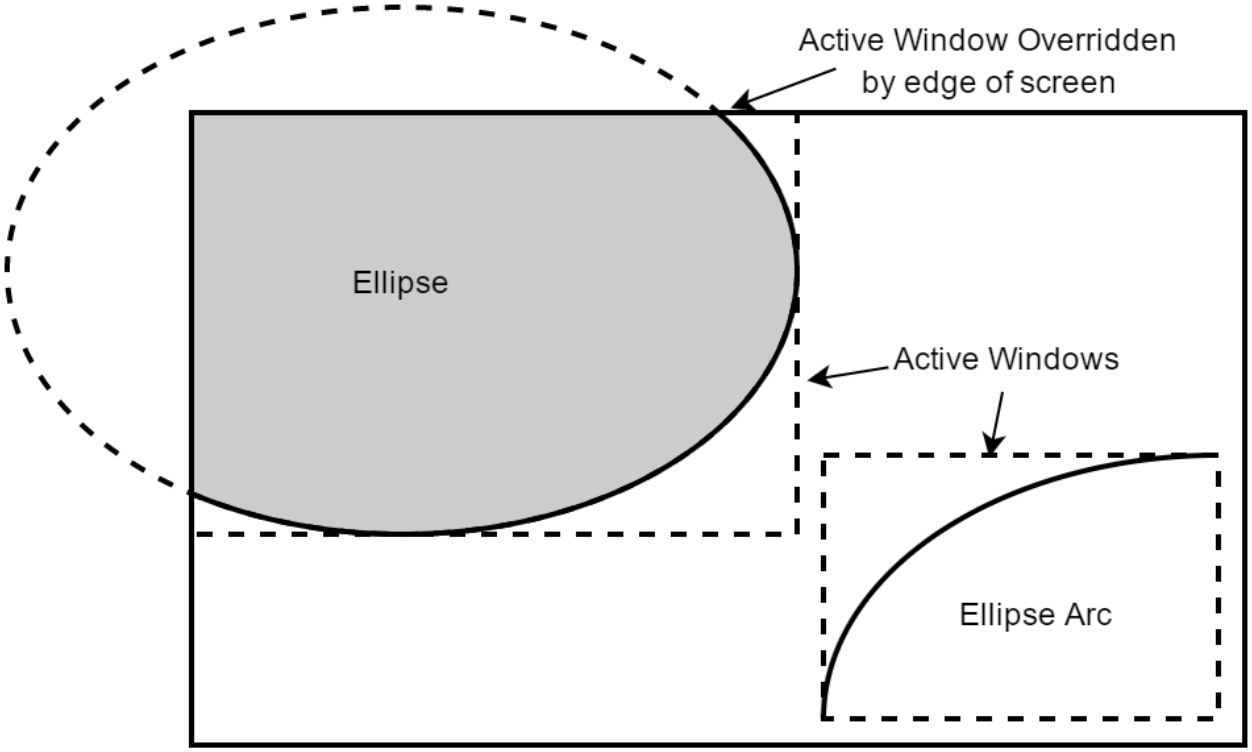


Figure 12: This diagram shows the ideal behaviour of an ellipse when it is drawn. Unfortunately, the Active Window is not supported for drawing ellipses, so there is not an elegant solution to this. The datasheet does not, however, state that the Active Window is not supported for the drawing of ellipse arcs, though this may simply be an oversight. If it is supported though, an ellipse can be drawn from a combination of four ellipse arcs, each of which could have a functional Active Window.

There are a few issues with the current usage of the 16-button keypad. Firstly, there is nothing to deal with the 'bouncing' that occurs when a button is pressed. This is the fact that when two electrical contacts are pushed into each other, it is often an elastic collision, and so the contacts rebound off each other. This means the button will rapidly connect and disconnect until it stabilises into the connected state. Therefore simple mechanical buttons and switches usually require a process known as 'debouncing' to eliminate this effect, and have the clean press the user intended. This project currently does not include any debouncing whatsoever, and instead simply tests whether the button is depressed or not. This means it is unable to distinguish two consecutive presses at the same mouse position from a single long press, and so the program is designed to ignore repeated presses at the same position for the geometric shapes. To take the triangle drawing as an example, while this is sufficient to prevent all three vertices of a triangle being set to the same point in three consecutive loops of the code, it also prevents the third vertex of one triangle being the same as the first vertex as the next, even if the user releases the 'click' button and repressed it a second time. This can be solved by sampling the button several times, and only registering the button as being pressed if the state has been the same for several samples in a row.

Secondly, while the code is able to differentiate any two different button presses by measuring the connections of each row in turn to the four columns, there are some three-button combinations that are indistinguishable from four buttons being pressed, as shown in Figure 13. As a result, the code will currently incorrectly state a button has been pressed when it has not if three or more buttons are simultaneously pressed. While it is impossible to distinguish these three button combinations from all four buttons being pressed, the code could be modified so that if four or more buttons are measured as being pressed, the code acts as if no buttons are being pressed. A useful extra feature would be to sound a buzzer in this case, to alert the user of the error.

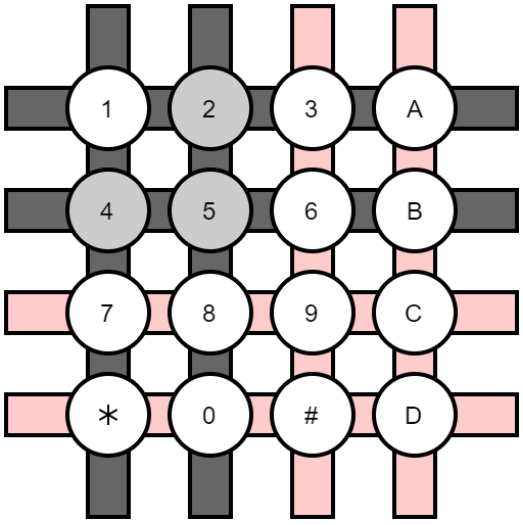


Figure 13: This diagram shows that even though the '1' button is not pressed, it will read as being pressed. If row one is driven low, then column two will be dragged low by the '2' button, so row 2 will be dragged low by the '5' button, and so column 1 will be dragged low by the '4' button. Therefore column 1 will be low when row one is low, so the program will interpret that as the '1' button being pressed, along with the '2', '4' and '5' buttons.

Finally, the 'Hexbutton' method which reads in the 16 button keypad state includes four 100µs delays. These are needed as the pull up resistors take some time to pull the inputs high, and so some delay is needed to ensure that has finished before reading the inputs. As Hexbutton is called three times per Main loop, this leads to a delay of approximately 1.2ms every loop, simply waiting for the pull up resistors to drift high. This can be improved in two ways. Firstly, instead of measuring the keypad state three times per loop, the keypad state could be measured once per loop, and the result saved to memory. Memory read operations are slow, taking two clock cycles to complete ([9] page 366), but they are still dramatically faster than a 100µs delay. With the processor running at 16MHz, a memory read operation will take a little over 100ns, almost 1000 times faster than the current implementation. Secondly, the 100µs delay itself is likely to be much greater than is necessary for the pull up resistors to become stable. By

combining both of these, the 1.2ms delay per loop could probably be reduced by at least a factor of 10, and so allow more paint operations per second.

However, the main loop currently has a fixed 15ms delay at the end anyway, to make the changes in loop time between different drawing modes smaller relative to the total loop time. This in turn means the speed of the mouse appears more uniform across different drawing modes. Unfortunately, this means that reducing the time spent on the keypad would effectively just shrink this fixed delay to about 14ms instead. To take advantage of improvements to the button pad, the loop time needs to be better controlled. This can be done by using one of the Timer/Counters on the Microprocessor, for example Timer/Counter0 complete ([9] page 92). By configuring the timer to trigger an interrupt after a time slightly longer than the longest loop time, a flag can be set to start the main loop again, and reset the timer. Therefore the main loop will take exactly as long as the timer to run, and so reducing the time spent on the button pad allows the timer period to be reduced, resulting in more loops per second, and so a smoother program.

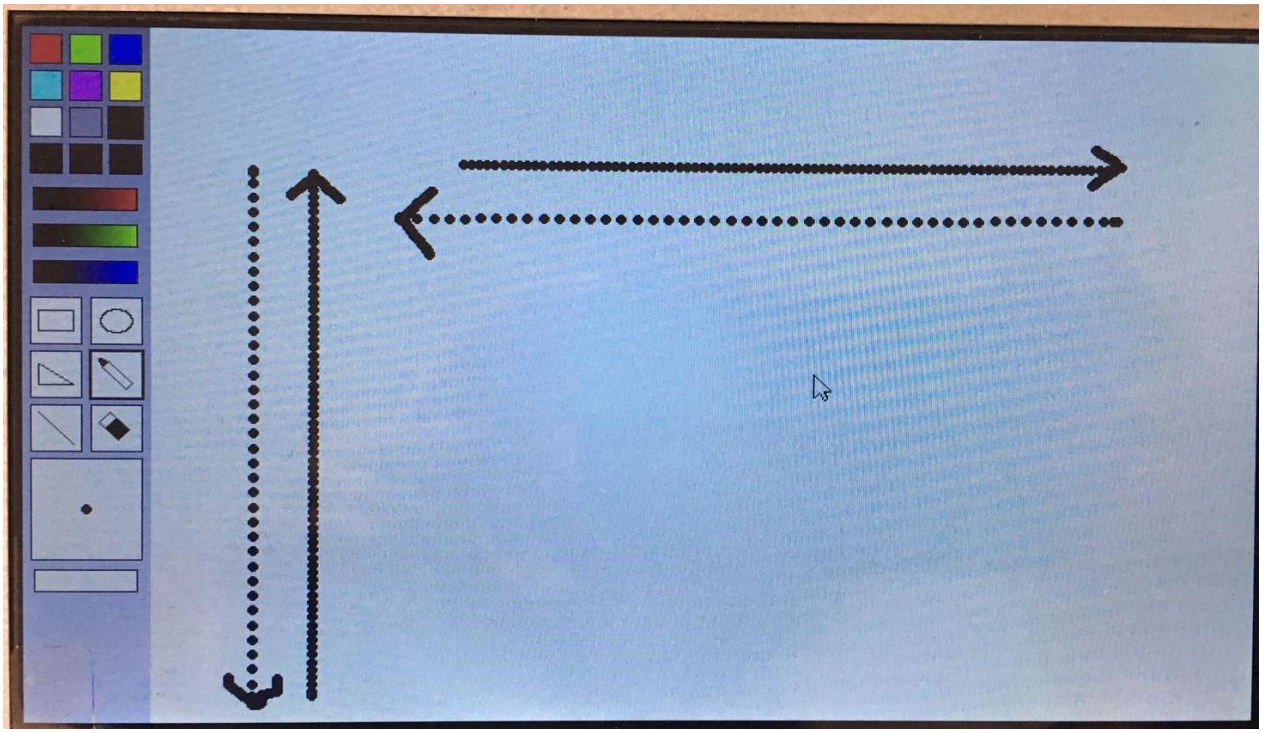


Figure 14: By holding the fast cursor button (the '3' button) and drawing with the Paint Brush tool, the difference in maximum speed in the positive and negative x directions and positive and negative y directions can be clearly seen. The cursor travels faster to the left, and faster down the screen, meaning there are larger spaces between each dot drawn than when travelling up or to the right.

There is also an issue with the joystick input, in that the centre position of the stick does not return a voltage in the middle of the range of results from the ADC. This results in a larger maximum possible displacement in the negative x direction than the positive x direction, and so the mouse moves faster in the negative x direction than the positive x direction, with the joystick pushed all the way to the left and right respectively. This can be visually shown in Figure 14, where the difference has been amplified by using holding the fast movement button as well as the draw button. While it is possible this is caused by non-linearity in the resistance of the potentiometer,

it is more likely that the ADC reference voltage on the Microprocessor is currently set to around 3.4V, while the voltage range on the potentiometer is 0-5V. If this is the cause of the problem, the solution would be to recalibrate the ADC reference voltage to 5V.

A second issue with the joystick is that sometimes the cursor will slowly drift downwards on the screen even when it is at the centre, the default position the joystick returns to when untouched by the user. This is because the range of values for which no cursor movement will occur, or 'deadzone' (currently the range of values with 10101 as their most significant five bits), is not well calibrated for the range of values the y-axis potentiometer can take at the default position. While the 'top five bits' test is a simple test, it is not necessarily aligned well with the actual range of values the potentiometer may take while at the default position. An improvement would be to also perform a subtraction before testing the top five bits, as this could effectively shift the deadzone to be centred on the average value of the default position.

Finally, a more mundane issue with the system is that it is very easy to accidentally disconnect the Driver from the Microprocessor by adjusting the screen position, as there is insufficient slack in the wires connecting the Driver and screen to the ProtoBoard. This can result in the Driver turning off, crashing, or even having data written to the wrong registers, depending on which pins become disconnected. While this issue has been somewhat mitigated by the use of a rubber band to help hold the connection wires in the correct position, it can be further reduced by using longer wires to increase the slack in the connection, and attaching the wires to the protoboard more rigidly to prevent them from simply sliding out when adjusted.

5. Modifications and Improvements

In addition to these fixes and improvements to existing features in the program, there are a few additional features that would complement the system nicely. The first of these would be to have a 'Colour Picker' tool. This would work in a very similar way to the current 'PaintPixel' method, which writes colour data directly to the relevant position in memory on the Driver. However, instead of writing to the address, the Colour Picker would read the data, and save it to one of the custom colours. According to the RA8875 data sheet, this would require an extra 'dummy' read command to be sent to the Driver, but would be straightforward to implement ([10] page 71).

Beyond this, the Driver supports text being displayed on screen by sending the ASCII code of each letter. Getting the program to the point of being able to display strings of characters on screen without having to set each pixel one by one would give the code much greater flexibility going forward.

Another modification that would improve the user experience would be to have the cursor icon change depending on the tool selected. The Driver supports having up to 8 32x32 pixel cursor icons saved at once, and they can be switched between by changing the value in register 0x41 on the Driver. The program currently only specifies one cursor icon, and it is set to a standard PC style mouse pointer. This means that all six tools currently in the program could have their own icon.

Finally, a minor UI improvement would be to either have the 3 custom colours appear visually distinct from the 9 fixed colours, for example being displayed in circles rather than squares, or for

all 12 colours to be customisable, with the with the primary, secondary and greyscale colours simply being the default values.

6. Conclusions

The aim of the project was to create a simple raster image editor and display, with a small toolset for a user to create images. This has been achieved, and the end product is a perfectly useable standalone art program. Figure 15 shows the result of a few minutes of effort using the program, demonstrating the use of some of the drawing tools. While there are a few issues with the system, as discussed in section 4, most are minor, and the proposed solutions should not take long to implement.

Figure 15: This photograph shows the result of a few minutes drawing in the final product. It depicts a simple boat in the ocean with a fish, on a bright, lightly clouded day. The image was created utilising all of the drawing tools with the exception of the rectangle tool.

However, at some level the actual art program is of secondary utility compared layers of interface between the RA8875 LCD Driver, and the ATMega128 microprocessor. This set of communications methods, and basic instruction set allows for a wide range of applications where a moderate resolution LCD display is of use, whether it is for displaying information from an experiment, or for a more casual application like a small video game.

The code used in the project is very much non-optimised, which means there are many performance improvements that could be applied to the specific application we have used it for, a real-time art program. However, this has some advantages if the small library of methods created for this project are reused in another application, as it means the code has not been optimised for a metric that is not relevant for this other application.

7. Product Specifications

This product is a standalone raster image display/editor with a 800x480 pixel LCD TFT display, comprising of a 720x480 pixel image area, or 'Canvas', and a 80x480 pixel GUI area. The product supports 16-bit colours, for over 65,000 different colours. This is achieved using an ATMega128 microprocessor to process inputs and store the state of the tools, a RA8875 LCD Driver Board to control the display and store the image data, and a 7.0-inch 40-pin TFT Display to display the image and GUI to the user.

The user can interact with the product using a Parallax 2-Axis joystick to move a cursor over the display, and a 16-button keypad to control the cursor speed, reset the canvas, and to activate, or 'click', the screen position under cursor. The cursor position is displayed using a desktop style mouse pointer icon. These allow the user to select a colour to draw in, and then draw on the canvas with one of six selectable tools.

The 80 pixels on the left-hand side of the screen are reserved for a GUI, containing clickable elements for nine fixed colours, three user-specified 'Custom Colours' and three colour sliders for the red, green and blue components of the custom colours respectively. Beneath these colour options, there are six clickable icons representing the drawing tools available to the user. Finally, beneath the tools, there is a 'Brush Size' indicator, and a slider to select the brush size.

The six drawing tools are only active on the canvas, and will not operate on the UI. The six drawing tools available to the user are as follows:

1. The Rectangle Tool. This draws a filled rectangle with opposing corners specified by two consecutive, different click positions on the screen. The rectangle is aligned with the pixel array of the screen.
2. The Ellipse Tool. This draws a filled ellipse centred on the first clicked point, and with the x and y semi-axes set by the x, y offset of the second point.
3. The Triangle Tool. This draws a filled triangle with vertices specified by three consecutive, different click positions on the screen.
4. The Paint Brush Tool. This draws a circle of the brush size centred on the clicked position on the screen.
5. The Line Tool. This draws a straight line between two consecutive, different click positions on the screen.
6. The Eraser Tool. This draws a circle of the brush size centred on the clicked position on the screen. It overrides the selected colour, and draws in white.

The RA8875 LCD Driver Board sends the state of the canvas and GUI to the display 60 times a second giving a smooth user experience. The input processing loop on the ATMega128 takes between 17ms and 20ms depending on the drawing operation being performed, giving an input sampling rate of more than 50Hz.

References

[1] Len Shustek, Computer History Museum (Jul 2010). *"MacPaint and QuickDraw SourceCode"*, http://www.computerhistory.org/atchm/macpaint-and-quickdraw-source-code/

[2] Cynthia Gregory Wilson, WWWiz Magazine (1995). *"Doug and Melody Wolfgram"*, http://wwwiz.com/issue01/wiz_c01.html
[3] Patrick Davison, Journal of Visual Culture (Dec 2014).*"Because of the Pixels: On the History, Form, and Influence of MS Paint"*, http://vcu.sagepub.com/content/13/3/275.full.pdf
[4] Rick Schieve and Gregg Woodcock (Mar 2002). *"Wells-Garnder Color Vector Monitor Guide"*, http://www.arcade-museum.com/manuals-monitors/FAQ%20Wells%20Gardner%206100%20Version%201.0%20dated%201%20Mar%2002.pdf
[5] Apple Inc (Oct 2016). *"iPad mini 2 with Retina display - Technical Specifications"*, https://support.apple.com/kb/sp693
[6] Tom Wright, IEEE (Apr 1998). *"History and Technology of Computer Fonts"*, http://ieeexplore.ieee.org/document/667294/
[7] 2-Axis Joystick, Parallax Inc. (2016). https://www.parallax.com/product/27800
[8] KP16 Switch Matrix Keypad, Industrologic, Inc. (2016). http://www.industrologic.com/kp16desc.htm
[9] ATMega128, Atmel Corporation (2016). http://www.atmel.com/images/doc2467.pdf
[10] RA8875 Driver Board, RAiO Technology Inc. (2012). https://cdnshop.adafruit.com/datasheets/RA8875_DS_V12_Eng.pdf
[11] 7.0" 40-pin TFT Display - 800x480 without Touchscreen, Adafruit (2016). https://www.adafruit.com/products/2353
[12] PB-203A Breadboard, Global Specialties (2016). http://www.globalspecialties.com/solderless-breadboards/breadboards-powered/item/89-pb-203.html
[13] Dr. Costas Foudas (2001). LCD1.asm Delay Methods, https://www.imperial.ac.uk/media/imperial-college/faculty-of-natural-sciences/department-of-physics/ug-labs/year-3/microprocessors/assembler-codes/LCD1.asm
[14] Limor Friend/Ladyada, K.Townsend/KTOWN, Adafruit Industries Adafruit Arduino (2016). *"Adafruit_RA8875.cpp"*, https://raw.githubusercontent.com/adafruit/Adafruit_RA8875/master/Adafruit_RA8875.cpp

Appendices

Appendix A, 16 Bit Colour

In physics, colour is often used synonymously with the wavelength of a beam of light. However, humans perceive the colour of a beam of light through its interaction with three different proteins in the eye, with each protein having a maximum response in the red, green and blue regions of the spectrum respectively. Therefore, as trying to recreate light with the same wavelength as an original image is technically very difficult, most digital displays will instead emit some red, some green, and some blue light, and attempt to match combined ocular response to all three to the ocular response to of the original frequency. Thus a physical 'colour' is stored and displayed as three separate values, commonly referred to as RGB values.

There are many ways of storing values for colour, but the RA8875 LCD Driver Board supports two formats, 8 bit and 16 bit colour. In both, the bits are split into three smaller values, representing the amount of red, green, and blue light in the colour. As neither 8 nor 16 is a multiple of 3, some colours will have more range than others. The human eye tends to have more green receptors than red or blue, so the green value is usually stored to higher precision. This means that in 8 bit colour, there are 3 bits of precision for red, 3 bits for green, and 2 for blue, creating a RRRGGGBB byte. As there are only 8 bits of data, this means there are only 256 possible colours in this format.

In this project we use 16 bit colour instead, which has 65,536 possible colours, which means the changes between similar colours can be much smaller, giving a much smoother appearance to colour gradients. In this format, there are 5 bits of precision for red data, 6 for green, and 5 for blue, creating a RRRRRGGGGGGBBBBB string of values.

Given data on the Driver and on the Microprocessor are both stored as separate bytes, this data gets split into two bytes, RRRRRGGG and GGGBBBBB, from which the separate RRRRR, GGGGGG and BBBBB values can be extracted using careful bit shifting.

Appendix B, Original Project Plan

Microprocessor Project Plan

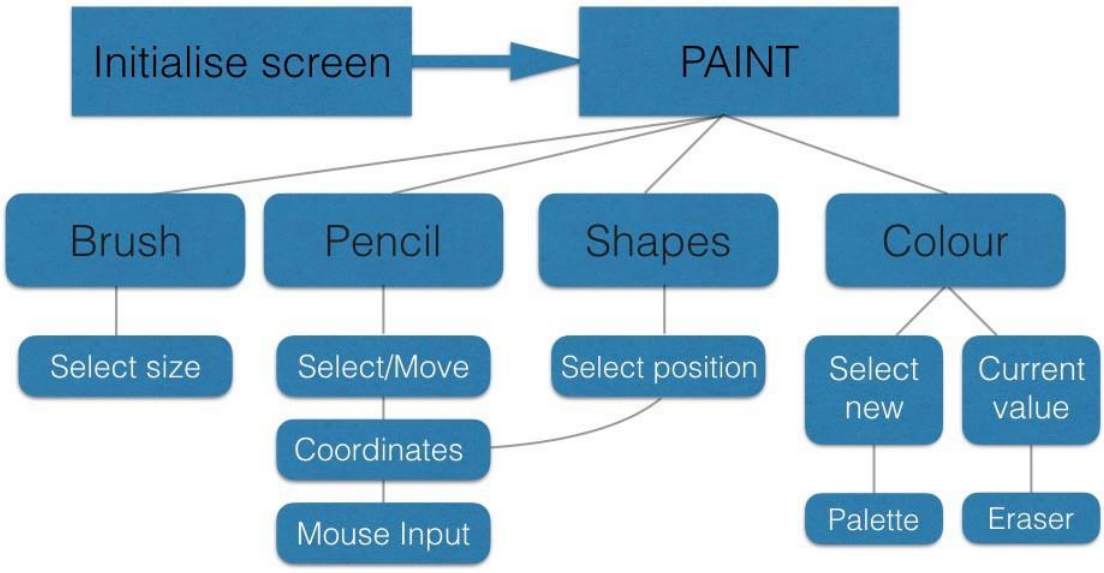
December 5, 2016

The goal is to build an MS-Paint style program using the ATmega128 microprocessor along with a driver board and an LCD screen. Two potentiometers will be used to select/move a position on the screen, acting as a mouse. A QWERTY keyboard will be used to perform actions such as "press", "reset", and "on/off".

Hardware

- ATMEL ATmega128 Microprocessor.
- RA8875 Driver Board.
- 7.0" 40-pin TFT Display (800x480).
- Voltmeters and keyboard.

Modular Design



Extension Tasks

- Reading back selected colour from image (i.e. colour dropper tool).
- Copy and paste.
- Text writing into Paint using keyboard.
- Other games (snake).

Division of Work

Joint:

- Interfacing with Driver.
- Program "Select" (e.g. Paint, Snake, etc)
- Keyboard Input and Potentiometers.

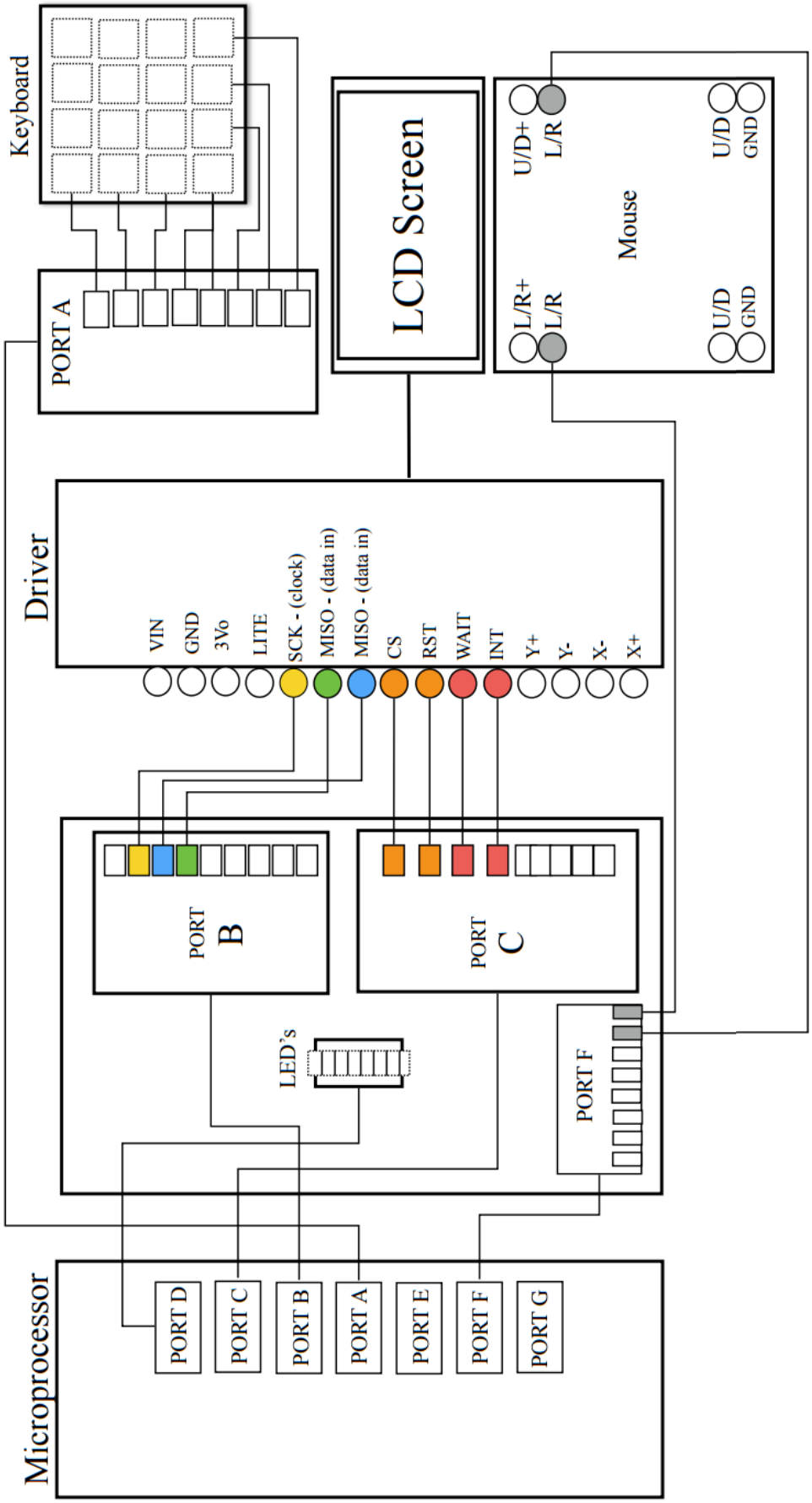
Maria:

- Colour
- Brush Size

Andrew:

- Pencil
- Geometric Shapes

Appendix C, Complete Hardware Diagram



Appendix D, Source Code

```
;
; main.asm
;
; Created: 24/11/2016 13:07:26
; Author : AL4413, MV914
;

.DEVICE ATmega128
.include "m128def.inc"

.ORG	$0

JMP Init

.include "Definitions.inc"
.include "DelayMethods.asm"
.include "SerialMethods.asm"
.include "GraphicalMethods.asm"
.include "LCDInitMethods.asm"
.include "CursorMethods.asm"
.include "UIMethods.asm"
.include "ToolMethods.asm"

MouseCursorData:					;Data string to set mouse cursor to windows style pointer.
		.DB
0b00101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b000010
10,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b00010010,0b1010
1010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b00010100,0b10101010,0b10
101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b00010101,0b00101010,0b10101010,0b
10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b00010101,0b01001010,0b10101010,0b10101010,
0b10101010,0b10101010,0b10101010,0b10101010,0b00010101,0b01010010,0b10101010,0b10101010,0b101010
10,0b10101010,0b10101010,0b10101010,0b00010101,0b01010100,0b10101010,0b10101010,0b10101010,0b1010
1010,0b10101010,0b10101010,0b00010101,0b01010101,0b00101010,0b10101010,0b10101010,0b10101010,0b10
101010,0b10101010,0b00010101,0b01010101,0b01001010,0b10101010,0b10101010,0b10101010,0b10101010,0b
10101010,0b00010101,0b01010101,0b01010010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,
0b00010101,0b01010101,0b01010100,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b000101
01,0b01010100,0b00000000,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b00010101,0b0001
0100,0b00101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b00010100,0b10000101,0b00
101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b00010010,0b10000101,0b00001010,0b
10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b00001010,0b10100001,0b01001010,0b10101010,
0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10100001,0b01001010,0b10101010,0b101010
10,0b10101010,0b10101010,0b10101010,0b10101010,0b10101000,0b00101010,0b10101010,0b10101010,0b1010
1010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10
101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b
10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,
0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b101010
10,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b1010
1010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10
101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b
10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,
0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b101010
10,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b1010
1010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10
```

```
101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b10101010,0b
10101010

;***** MEMORY ADDRESSES FOR PROGRAM VARIABLES *****
;                                CursorPosition   0110,0111,0112,0113
;             PreviousMousePosition  0114,0115,0116,0117
;                         LargeMousePosition       0118,0119,011A,011B
; PreviousLargeMousePosition       011C,011D,011E,011F
;                                ForegroundColour          0120,0121
;                       TempForegroundColour    0122,0123
;                        CurrentDrawingMode     0125
;             RectangleDrawingState  0130
;           RectangleDrawingPosition 0131,0132,0133,0134
;                         EllipseDrawingState       0135
;            EllipseDrawingPosition    0136,0137,0138,0139
;                         TriangleDrawingState      0140
;           TriangleDrawingPosition1  0141,0142,0143,0144
;           TriangleDrawingPosition2  0145,0146,0147,0148
;                                LineDrawingState0150
;                         LineDrawingPosition       0151,0152,0153,0154
;                                          BrushSize        0155
;                         ColourSquareSelected      0156
;                                ColourSlider       0157,0158,0159
;                                CustomColours 0160,0161, 0162,0163, 0164,0165
;MouseSpeed smaller = faster       0170



Init:
                ;Stack Pointer Setup Code
                ldi r16, $0F                                                    ;Stack Pointer Setup to 0x0FFF
                out SPH,r16                                                             ;Stack Pointer High Byte
                ldi r16, $FF                                                    ;Stack Pointer Setup
                out SPL,r16                                                             ;Stack Pointer Low Byte

                ;RAMPZ Setup -  1 = EPLM acts on upper 64K, 0 = EPLM acts on lower 64K
                ldi  r16, $00                                                   ;Lower memory page arithmetic
                out RAMPZ, r16


                ;Port A does not need to be set up here, as it will be set in the HexButton method when reading
which key has been pressed.
                ;Port B does not need to be set up here, as it will be set in the SPI_MasterInit method later.

                ;Port C Setup. Lowest two bits as Outputs (Chip Select and RESET pins to LCD Driver), and rest as
inputs (WAIT,INTERUPT pins from LCD Driver)
                ldi r16, $03                                                    ;Lowest two bits as Outputs. Upper
6 bits as Inputs.
                out DDRC , r16                                                  ;Set Port C Direction Register to
value in r16
                ldi r16, $FC                                                    ;Enable pull up resistors on input
pins to prevent current spikes from high impedence input. Set initial values of output pins to zero.
                out PORTC, r16                                                  ;Enable pull up resistors and set
initial Port C values.

                ;Port D Setup. Set as Outputs, and connected to LEDs for debug purposes.
```

```
		ldi r16, $ff			;Set all 8 pins as outputs.
		out DDRD , r16			;Set Port D Direction Register to
value in r16
		ldi r16, $00			;Initial Port D value = $00.
		out PORTD, r16			;Output $00 to Port D.

		;Port E is unused.

		;Port F Setup. Set as inputs for connecting to joystick potentiometers.
		ldi r16, $00			;Set all 8 pins as inputs.
		STS DDRF, r16			;PSet Port F Direction Register to
value in r16
		ldi r16, $FF			;Enable pull up resistors on all pins
to prevent current spikes from high impedence input.
		STS PORTF, r16			;Enable pull up resistors.

		call SPI_MasterInit		;Configure device as master, and
set clock rate.

		;Reset LCD Driver, to clear any existing settings
		ldi r16,$00
		out PORTC, r16			;drive the Chip Select and RESET
pins on the LCD Driver low (active)
		call DEL49ms			;wait, to let the Driver reset.
		ldi r16,$03
		out PORTC, r16			;drive the Chip Select and RESET
pins on the LCD Driver high (inactive)

		;Configures the Analog to Digital Converter
		ldi r16, $83			;ADC Interrupt Disabled, ADC
Enable
		out ADCSR, r16			;ADC single shot Mode, Prescaler:
CK/8

		call LCDTurnOn			;Sets configuration values for LCD
screen, including screen size, and VSync/HSync setting.
		call SetCursorShape		;Write Mouse pointer image data to
Driver.
		call PaintReset			;Enable LCD screen, reset Tools,
reset Custom Colours, set selected colour to black and tool to paintbrush, move mouse to center of screen, draw UI

		rjmp Main				;Begin main program loop.

;The main program loop. Everything within the program after initialisation is run from here.
Main:
		call SpeedModeTest		;Sets the MouseSpeed variable
depending on whether the fast or slow motion button are pressed.
		call MouseMove			;Moves the mouse according to the
position of the joystick.
		call CursorDisplay		;Send the current mouse position
to the Driver.

		;check if we are clicking:
		call HexButton			;read button code from hexbutton
board into r17
```

```
                SBRS r17, 7                                     ;if the button code corresponds to
button '1'...
                rjmp MainEnd                                    ;...then skip this line, and do not
run any of the 'mouse click' methods. This reduces the number of operations the program has to do per loop.

                call Paint                                      ;If clicking on canvas, do
drawing operations

                call SelectColours                              ;If clicking on Colour Squares, set selected
colour.

                call SelectTool1                                ;If clicking on Tool icons, set selected tool,
and reset.
                call SelectTool2
                call SelectTool3
                call SelectTool4
                call SelectTool5
                call SelectTool6

                call SelectRedSliderColour              ;If clicking on colour sliders, set custom colour to new
colour.
                call SelectGreenSliderColour
                call SelectBlueSliderColour

                call SelectBrushSliderSize              ;If clicking on Brush size slider, set brush size.

        MainEnd:
                ;check if reset button is pressed:
                call HexButton                                  ;Read button code from hexbutton
board into r17
                SBRC r18, 0                                     ;If reset button ('D')
pressed, then call PaintReset
                rcall PaintReset                                ;Reset Tools, reset Custom Colours, set
selected colour to black and tool to paintbrush, move mouse to center of screen, redraw UI.

                call DEL15ms

                rjmp Main


SpeedModeTest:                                                  ;Sets the MouseSpeed variable
depending on whether the fast or slow motion button are pressed.
                push r16
                push r17
                push r18
                push ZL
                push ZH

                ldi ZL,$70                                      ;Load address for the
MouseSpeed into Z register.
                ldi ZH,$01
                ldi r16,3                                       ;divide joystick offset by 2^3 = 8
before applying to mouse position
                ST Z,r16                                        ;Set MouseSpeed = 3 (default)

                ;check if slow-mo button is pressed:
```

```
                call HexButton                          ;read button code from hexbutton
board into r17
                SBRC r17, 4                                     ;if pressing 'A' key...
                rjmp SpeedModeTestSlow                  ;...set speed to Slow
                SBRC r17,5                                      ;if pressing '3' key...
                rjmp SpeedModeTestFast                  ;...set speed to Fast.

                rjmp SpeedModeTestEnd                   ;If pressing neither, leave speed at default.

        SpeedModeTestSlow:
                ldi r16,5                                       ;divide joystick offset by 2^5 = 32
before applying to mouse position
                ST Z,r16                                        ;Set MouseSpeed = 5 (slow)

                jmp SpeedModeTestEnd                    ;pop back registers and return.

        SpeedModeTestFast:
                ldi r16,1                                       ;divide joystick offset by 2^1 = 2
before applying to mouse position
                ST Z,r16                                        ;Set MouseSpeed = 1 (fast)

        SpeedModeTestEnd:                               ;pop back registers and return.
                pop ZH
                pop ZL
                pop r18
                pop r17
                pop r16
                ret

MouseMove:                                                      ;Moves the mouse
according to the position of the joystick.
                push r16
                push r17
                push r18
                push r19
                push r20
                push r21
                push r22
                push r23
                push r24
                push YL
                push YH
                push ZL
                push ZH

                ;Copy mouse position to PreviousMousePosition
                ldi YL,$10                                      ;Load address for the
CursorPosition into Y register.
                ldi YH,$01
                LD r16, Y+                                      ;Load old CursorPosition
into r16-r19
                LD r17, Y+
                LD r18, Y+
                LD r19, Y+
                ST Y+, r16                                      ;Store old CursorPosition
into PreviousMousePosition memory.
```

```
                ST Y+, r17
                ST Y+, r18
                ST Y+, r19

                ;Copy large mouse position to PreviousLargeMousePosition
                ldi YL,$18                      ;Load address for the
LargeMousePosition into Y register.
                ldi YH,$01
                LD r16, Y+                      ;Load old
LargeMousePosition into r16-r19
                LD r17, Y+
                LD r18, Y+
                LD r19, Y+
                ST Y+, r16                      ;Store old
LargeMousePosition into PreviousLargeMousePosition memory.
                ST Y+, r17
                ST Y+, r18
                ST Y+, r19

                ;get joystick xpos
                ldi r18,$00
                call ADCsel                     ;select X potentiometer for
DCATrig
                call DCATrig                    ;returns result of ADC in r19/r20

                lsr r20                         ;divide potentiometer
value by 32 -> range from 0 to 31
                ror r19                         ;so we can see if the
joystick is in the centre.
                lsr r20
                ror r19
                lsr r19
                lsr r19
                lsr r19

                cpi r19,21                      ;if joystick inside deadzone
(central 1/32 of the range), jump to trying the y axis (i.e. don't change the x position). 21 = measured centre x
position (691) / 32.
                BREQ MouseMoveY

                ldi YL,$18                      ;Load address for the
LargeMousePosition x component into Y register.
                ldi YH,$01
                LD ZL,Y+                        ;load large mouse x position into Z
register
                LD ZH,Y+

                call DCATrig                    ;Returns result of ADC in r19/r20,
i.e. the x value of the joystick.

                ldi r17,$B3                     ;Load measured centre x
position (691) into r17/r18.
                ldi r18,$02

                ldi YL,$70                      ;Load address for the
MouseSpeed into Y register.
```

```
                ldi YH,$01
                ld r25,Y                        ;Load MouseSpeed into r25
                ldi r26,0                       ;Set loopCounter to zero.
        MouseMoveXLoop:                         ;Divides changes to Mouse position
by 2 as many times as MouseSpeed.
                        subi r26,$ff            ;add 1 to loop counter
                        lsr r18                         ;divides joystick centre
position by 2
                        ror r17
                        lsr r20                         ;divides joystick position
by 2
                        ror r19
                        cpse r26,r25            ;If divided by 2 as many times as
MouseSpeed, continue, else loop again.
                        rjmp MouseMoveXLoop

                add ZL,r19                              ;add joystick value to new
large mouse position.
                adc ZH,r20
                sub ZL,r17                              ;subtract centre joystick
value from new large mouse position.
                sbc ZH,r18

                ldi YL,$18                              ;Load address for the
LargeMousePosition x component into Y register.
                ldi YH,$01
                ST Y+, ZL                       ;Update LargeMousePosition x
component with new value.
                ST Y+, ZH

                lsr ZH                                  ;Divide new
LargeMousePosition by 32
                ror ZL
                lsr ZH
                ror ZL
                lsr ZH
                ror ZL
                lsr ZH
                ror ZL
                lsr ZH
                ror ZL

                ldi YL,$10                              ;Load address for the
CursorPosition x component into Y register.
                ldi YH,$01
                ST Y+, ZL                       ;Update CursorPosition x
component with new value.
                ST Y+, ZH

                call MouseOnScreen              ;If Mouse on screen, return $FF into
r16, else return $00
                cpi r16,$ff
                BREQ MouseMoveY                         ;If mouse still on screen
after changing x position, continue to changing y position...
                                                        ;...else reset
mouse position to old mouse position:
```

```
                ;if mouse off-screen, reset to old mouse position.
                ldi YL,$14                      ;Load address for the
PreviousMousePosition x component into Y register.
                ldi YH,$01
                LD ZL,Y+                        ;Load PreviousMousePosition x
component into Z register
                LD ZH,Y+

                ldi YL,$10                      ;Load address for the
CursorPosition x component into Y register.
                ldi YH,$01
                ST Y+, ZL                       ;Update CursorPosition x
component with old value.
                ST Y+, ZH

                ldi YL,$1C                      ;Load address for the
PreviousLargeMousePosition x component into Y register.
                ldi YH,$01
                LD ZL,Y+                        ;Load PreviousLargeMousePosition
x component into Z register.
                LD ZH,Y+

                ldi YL,$18                      ;Load address for the
LargeMousePosition x component into Y register.
                ldi YH,$01
                ST Y+, ZL                       ;Update LargeMousePosition x
component with old value.
                ST Y+, ZH

        MouseMoveY:
                ;get joystick ypos
                ldi r18,$01
                call ADCsel                     ;select Y potentiometer for
DCATrig
                call DCATrig                    ;Returns result of ADC in r19/r20,
i.e. the y value of the joystick.

                lsr r20                         ;divide potentiometer
value by 32 -> range from 0 to 31
                ror r19                         ;so we can see if the
joystick is in the centre.
                lsr r20
                ror r19
                lsr r19
                lsr r19
                lsr r19

                cpi r19,21                      ;if joystick inside deadzone
(central 1/32 of the range), jump to the end (i.e. don't change the y position). 21 = measured centre x position (688)
/ 32.
                BREQ MouseMoveEnd

                ldi YL,$1A                      ;Load address for the
LargeMousePosition y component into Y register.
                ldi YH,$01
```

```
                LD ZL,Y+                                ;Load large mouse y position into Z
register.
                LD ZH,Y+

                call DCATrig                            ;Returns result of ADC in r19/r20,
i.e. the y value of the joystick.

                ldi r17,$b0                                     ;Load measured centre x
position (688) into r17/r18.
                ldi r18,$02

                ldi YL,$70                                      ;Load address for the
MouseSpeed into Y register.
                ldi YH,$01
                ld r25,Y                                ;Load MouseSpeed into r25
                ldi r26,0                               ;Set loopCounter to zero.
                MouseMoveYLoop:                                 ;Divides changes to Mouse
position by 2 as many times as MouseSpeed.
                        subi r26,$ff                    ;add 1 to loop counter
                        lsr r18                                 ;divides joystick centre
position by 2
                        ror r17
                        lsr r20                                 ;divides joystick position
by 2
                        ror r19
                        cpse r26,r25                    ;If divided by 2 as many times as
MouseSpeed, continue, else loop again.
                        rjmp MouseMoveYLoop

                ;operations backwards on y, as y position measured from top of screen, not bottom
                sub ZL,r19                                      ;subtract joystick value
from new large mouse position.
                sbc ZH,r20
                add ZL,r17                                      ;add centre joystick value
to new large mouse position.
                adc ZH,r18

                ldi YL,$1A                                      ;Load address for the
LargeMousePosition y component into Y register.
                ldi YH,$01
                ST Y+, ZL                               ;Update LargeMousePosition y
component with new value.
                ST Y+, ZH

                lsr ZH                                  ;Divide new
LargeMousePosition by 32
                ror ZL
                lsr ZH
                ror ZL
                lsr ZH
                ror ZL
                lsr ZH
                ror ZL
                lsr ZH
                ror ZL
```

```
                ldi YL,$12                      ;Load address for the
CursorPosition y component into Y register.
                ldi YH,$01
                ST Y+, ZL                       ;Update CursorPosition y
component with new value.
                ST Y+, ZH

                call MouseOnScreen              ;If Mouse on screen, return $FF into
r16, else return $00
                cpi r16,$ff
                BREQ MouseMoveEnd               ;If mouse still on screen after
changing y position, continue to end...
                                                ;...else reset
mouse position to old mouse position:

                ;if mouse off-screen, reset to old mouse position.
                ldi YL,$16                      ;Load address for the
PreviousMousePosition y component into Y register.
                ldi YH,$01
                LD ZL,Y+                        ;Load PreviousMousePosition y
component into Z register
                LD ZH,Y+

                ldi YL,$12                      ;Load address for the
CursorPosition y component into Y register.
                ldi YH,$01
                ST Y+, ZL                       ;Update CursorPosition y
component with old value.
                ST Y+, ZH

                ldi YL,$1E                      ;Load address for the
PreviousLargeMousePosition y component into Y register.
                ldi YH,$01
                LD ZL,Y+                        ;Load PreviousLargeMousePosition
y component into Z register.
                LD ZH,Y+

                ldi YL,$1A                      ;Load address for the
LargeMousePosition y component into Y register.
                ldi YH,$01
                ST Y+, ZL                       ;Update LargeMousePosition y
component with old value.
                ST Y+, ZH

        MouseMoveEnd:
                pop ZH
                pop ZL
                pop YH
                pop YL
                pop r24
                pop r23
                pop r22
                pop r21
                pop r20
                pop r19
                pop r18
```

```
                pop r17
                pop r16
                ret

PaintReset:                                                     ;Enable LCD screen, reset
Tools, reset Custom Colours, set selected colour to black and tool to paintbrush, move mouse to center of screen,
draw UI
                push r16
                push r25
                push r26
                push YL
                push YH

                rcall NewScreen                                 ;Enable LCD screen, turns on the
backlight, and sets every pixel white.

                ldi YL,$10                                      ;Load address of the
CursorPosition into Y register.
                ldi YH,$01
                ldi r16,$90                                     ;xlow
                st Y+,r16
                ldi r16,$01                                     ;xhigh
                st Y+,r16                                       ;Set x value of CursorPosition to
$0190 = 400, i.e. halfway across screen.

                ldi r16,$f0                                     ;ylow
                st Y+,r16
                ldi r16,$00                                     ;yhigh
                st Y+,r16                                       ;Set y value of CursorPosition to
$00F0 = 240 , i.e. halfway down screen.

                ldi YL,$18                                      ;Load address of the
LargeMousePosition into Y register.
                ldi YH,$01
                ldi r16,$00                                     ;xlow
                st Y+,r16
                ldi r16,$32                                     ;xhigh
                st Y+,r16                                       ;Set x value of LargeMousePosition
to $3200 = 12800 = 400*32, i.e. halfway across screen.

                ldi r16,$00                                     ;ylow cursor
                st Y+,r16
                ldi r16,$1E                                     ;yhigh cursor
                st Y+,r16                                       ;Set y value of LargeMousePosition
to $1E00 = 7860 = 240*32 , i.e. halfway down screen.

                call ResetTools                                 ;Clear any saved points for drawing
tools.
                call ResetBrush
                call CustomColoursReset                 ;Reset custom colours to black.

                ldi YL,$25                                      ;Load address of
CurrentDrawingMode into Y register.
                ldi YH,$01
                ldi r16,$00
```

```
                ST Y,r16                                ;Set the CurrentDrawingMode to 0
(Paint Brush).

                ldi YL,$56                              ;Load address of the
ColourSquareSelected into Y register.
                ldi YH,$01
                ldi r16,9
                ST Y,r16                                ;Set the ColourSquareSelected to 9
(Black).

                ldi r25, COLOUR_BLACKL
                ldi r26, COLOUR_BLACKH
                call SetForegroundColour        ;Set Initial ForegroundColour to Black, so user paints
in black

                call DrawUserInterface          ;Redraw the user interface. This happens
after the custom colours are reset and the foreground colour changed, so the custom colour icons display correctly,
and the brush size icon has the correct center colour.

                call DEL49ms                            ;A couple of delays to reduce the
flickering that occurs when the Reset button is held on the button pad.
                call DEL49ms

                pop YH
                pop YL
                pop r26
                pop r25
                pop r16
                ret

NewScreen:                                              ;Enable LCD screen, turns
on the backlight, and sets every pixel white.
                call SaveForegroundColourToTemp ;Backup the current foreground colour, so we can
draw in another colour until the end of the method.

                ldi r16, $ff                            ;Load $ff into r16 as boolean TRUE
argument for DisplayOn method.
                call DisplayOn                          ;Turn the LCD display on.

                ldi r16, $ff
                call GPIOX                              ;Enable TFT - display
enable tied to GPIOX

                ldi r16, $ff                            ;Load $ff into r16 as boolean TRUE
argument for PWM1config method. I.e. turn on the backlight.
                ldi r17, $0A                            ;set clock rate as 2^10 times slower
than the Driver system clock.
                call PWM1config                         ;Turn on the backlight, and set the
power management clock rate to 1024 times slower than driver system clock.

                ldi r16, $3f
                call PWM1out
                ldi r25, COLOUR_WHITEL          ;Load colour white into r25/26.
                ldi r26, COLOUR_WHITEH
                call SetForegroundColour        ;Set foreground colour to white.
```

```
		call FillScreen					;Draw a white rectangle over the
entire screen.

		call LoadForegroundColourFromTemp	;Reset foreground colour to saved colour, so the
foreground colour has not been altered from outside this method.
		ret

HexButton:							;Returns 2 bytes r17/r18,
containing state of all 16 buttons on number pad. 1 = pressed, 0 = unpressed.
		push r16
		push r19

		;GetButtons 123A
		ldi r16, $08
		out DDRA, r16					;Set pin 3 as an output, and all
other pins as inputs.
		ldi r16, $F7
		out PORTA, r16					;Drive pin 3 low, and enable pull up
resistors on all other pins. If any buttons on row 0 are pressed, they connect a pull up to low, and low drags the pull
up resistor low.
		call DEL100mus					;Delay a small amount so any
capacitance in the button pad is filled before trying to read in the state.
		In r17, PINA
		ori r17,$0f					;Throw away any
connections that are not columns
		com r17						;invert, so 1 is pressed, and
0 is unpressed

		;GetButtons 456B
		ldi r16, $04
		out DDRA, r16					;Set pin 2 as an output, and all
other pins as inputs.
		ldi r16, $FB
		out PORTA, r16					;Drive pin 2 low, and enable pull up
resistors on all other pins. If any buttons on row 2 are pressed, they connect a pull up to low, and low drags the pull
up resistor low.
		call DEL100mus					;Delay a small amount so any
capacitance in the button pad is filled before trying to read in the state.
		In r18, PINA
		ori r18,$0f					;Throw away any
connections that are not columns
		com r18						;invert, so 1 is pressed, and
0 is unpressed
		lsr r18						;shift right, so line 2 state
in bits [3:0]
		lsr r18
		lsr r18
		lsr r18
		or r17,r18					;Merge states of buttons
123A456B into a single byte


		;GetButtons 789C
		ldi r16, $02
		out DDRA, r16					;Set pin 1 as an output, and all
other pins as inputs.
```

```
		ldi r16, $FD
		out PORTA, r16		;Drive pin 1 low, and enable pull up
resistors on all other pins. If any buttons on row 3 are pressed, they connect a pull up to low, and low drags the pull
up resistor low.
		call DEL100mus		;Delay a small amount so any
capacitance in the button pad is filled before trying to read in the state.
		In r18, PINA
		ori r18,$0f			;Throw away any
connections that are not columns
		com r18				;invert, so 1 is pressed, and
0 is unpressed

		;GetButtons *0#D
		ldi r16, $01
		out DDRA, r16		;Set pin 0 as an output, and all
other pins as inputs.
		ldi r16, $FE
		out PORTA, r16		;Drive pin 0 low, and enable pull up
resistors on all other pins. If any buttons on row 4 are pressed, they connect a pull up to low, and low drags the pull
up resistor low.
		call DEL100mus		;Delay a small amount so any
capacitance in the button pad is filled before trying to read in the state.
		In r19, PINA
		ori r19,$0f			;Throw away any
connections that are not columns
		com r19				;invert, so 1 is pressed, and
0 is unpressed
		lsr r19				;shift right, so line 4 state
in bits [3:0]
		lsr r19
		lsr r19
		lsr r19
		or r18,r19
		out portD,r18		;Merge states of buttons 789C*0#D
into a single byte

		pop r19
		pop r16
		ret

ADCsel:					;Select channel based on
value in r18, i.e which pin in PORTF to pass to the ADC.
		out ADMUX, r18		;Tell the multiplexer which input to
output to the ADC.
		ret

DCATrig:				;Returns result of Analogue to
Digital Conversion (ADC) in r19/r20
		SBI ADCSR, 6		;Set the 'ADC Start Conversion' bit
in the ADC Status Register (ADCSR), making it convert the voltage on the selected PORTF pin to a 10 bit digital
number
	DCATrigLoop:
		SBIS ADCSR, 4		;If ADC Interrupt flag is not high...
		RJMP DCATrigLoop	;...loop until ADC Interrupt flag is
high.
```

```
                SBI ADCSR, 4                                    ;...else reset ADC Interrupt flag and
continue.
                IN r19, ADCL                                    ;Move the low byte of ADC result
into r19
                IN r20, ADCH                                    ; Read in High Byte
                RET


;###########################################################################
; Definitions.inc
;
; Created: 24/11/2016 13:09:26
; Authors : AL4413, MV914
;


;16 Bit Colours
#define  COLOUR_BLACKL      0x00
#define  COLOUR_BLACKH      0x00
#define COLOUR_GREYL        0x18        ;75% grey
#define COLOUR_GREYH        0xc6
#define COLOUR_WHITEL       0xFF
#define COLOUR_WHITEH       0xFF
#define  COLOUR_REDL        0x00
#define  COLOUR_REDH        0xF8
#define  COLOUR_GREENL      0xE0
#define  COLOUR_GREENH      0x07
#define  COLOUR_BLUEL       0x1F
#define  COLOUR_BLUEH       0x00
#define COLOUR_CYANL        0xFF
#define COLOUR_CYANH        0x07
#define COLOUR_MAGENTAL     0x1F
#define COLOUR_MAGENTAH     0xF8
#define COLOUR_YELLOWL      0xE0
#define COLOUR_YELLOWH      0xFF

; DelayMethods.asm
; File contains code written by Dr Costas Foudas, as below.
;
; Name : LCD1.asm Delay Methods
; Author : Dr Costas Foudas, Imperial College, High Energy Physics division
; Date : 2001
; Available : https://www.imperial.ac.uk/media/imperial-college/faculty-of-natural-sciences/department-of-
physics/ug-labs/year-3/microprocessors/assembler-codes/LCD1.asm

;*******************   DELAY ROUTINES ********************************************
BigDEL:
        rcall Del49ms
        rcall Del49ms
        rcall Del49ms
        rcall Del49ms
        rcall Del49ms
        ret
;
DEL15ms:
;
```

```
; This is a 15 msec delay routine. Each cycle costs
; rcall         -> 3 CC
; ret           -> 4 CC
; 2*LDI       -> 2 CC
; SBIW        -> 2 CC * 19997
; BRNE       -> 1/2 CC * 19997
;

        LDI XH, HIGH(19997)
        LDI XL, LOW (19997)
          COUNT:
        SBIW XL, 1
        BRNE COUNT
        RET
;
DEL4P1ms:
        LDI XH, HIGH(5464)
        LDI XL, LOW (5464)
          COUNT1:
        SBIW XL, 1
        BRNE COUNT1
        RET
;
DEL100mus:
        LDI XH, HIGH(131)
        LDI XL, LOW (131)
          COUNT2:
        SBIW XL, 1
        BRNE COUNT2
        RET
;
DEL49ms:
        LDI XH, HIGH(65535)
        LDI XL, LOW (65535)
          COUNT3:
        SBIW XL, 1
        BRNE COUNT3
        RET

;
; SerialMethods.asm
;
; Created: 24/11/2016 13:11:57
; Authors: AL4413, MV914
;
; File also contains code written by Prof. Mark Neil, Imperial College. Accreditation is commented below.


SerialTestLoop:                         ;Debug method - writes $ff to register $63 on the driver (foreground
colour low byte), and then reads it back. Used with oscilliscope on MISO/MOSI pins to see data transfer.
                call SerialTestWrite

                call SerialTestRead             ;read value back from register 63 on driver

                call DEL100mus
                call DEL100mus
```

```
                call DEL100mus
                call DEL100mus
                call DEL100mus
                call DEL100mus
                call DEL100mus
                call DEL100mus
                jmp SerialTestLoop

SerialTestWrite:                ;Debug method - writes $ff to register $63 on the driver (foreground
colour low byte)
                ldi r17,$63
                ldi r18,$ff
                call WriteReg
                ret

SerialTestRead:                 ;Debug method - reads value of register $63 back from driver
and outputs it to Port D.
                ldi r17,$63
                call ReadReg
                out portd,r17
                ret

; Name : SPI_MasterInit and SPI_MasterTransmit methods
; Author : Prof. Mark Neil, Imperial College, Photonics Group
; Date : 2016
; Available : https://www.imperial.ac.uk/media/imperial-college/faculty-of-natural-sciences/department-of-
physics/ug-labs/year-3/microprocessors/6_Serial_Parallel.pdf
SPI_MasterInit:                 ;Configure device as master, and set clock rate. Set I/O
directions on portB as required for 4 wire SPI
                push r17

                ;Set MOSI, Serial Clock (SCK), SS*  as outputs, all others inputs
                ldi r17,(1<<DDB2)|(1<<DDB1)|(1<<DDB0)
                out DDRB,r17

                ; Enable SPI, Configure this device as Master, set clock rate fck/16
                ldi r17,(1<<SPE)|(1<<MSTR)|(1<<SPR0)
                out SPCR,r17

                pop r17
                ret

SPI_MasterTransmit:             ;Send data from r16 to the Driver via SPI.
                out SPDR,r16            ;Start transmission of data (held in r16)
        Wait_Transmit:          ;Wait for transmission to complete
                sbis SPSR,SPIF          ;If transmission complete, return, else wait more.
                rjmp Wait_Transmit
                ret

DriverWait:                     ;Wait until WAIT pin from driver is cleared. Used to
make sure instructions are not sent to the driver while it is busy.
                push r16

        DriverWaitStart:
                IN r16, PINC            ;input from port C, to check INT and WAIT pins
                ANDI r16, $04           ;AND with 00000100, to isolate wait pin from input
```

```
		CPI r16, $00			;compare with $00, to see if WAIT pin is set. If WAIT pin set,
loop until WAIT pin cleared.
		BREQ DriverWaitStart	;If WAIT pin = 1, jump back to start of loop else finish, and return.

	DriverWaitEnd:
		pop r16
		ret

DriverDataR:				;read data from driver into r17 from register address specified
by previous DriverCommandW.
		push r16

		ldi r16,$02			;chip select - lcd input active. ($02 so the Chip Select
pin is driven low (active), while the RESET pin is kept high (inactive).)
		out PORTc,r16

		ldi r16,$40			;Writes 0b01(000000) to driver, so it knows to
transmit data from register on the next SCK
		rcall SPI_MasterTransmit

		ldi r16,$00			;dummy data transmit, to fill SPDR register with data
from driver (SPI works with full Duplex)
		rcall SPI_MasterTransmit
		in r17, SPDR

		ldi r16,$03			;chip select - lcd inactive. ($03 so the Chip Select pin
is driven high (inactive), while the RESET pin is kept high (inactive).)
		out PORTc,r16

		pop r16
		ret

DriverDataW:				;write data in r17 to the register on the driver with address
specified by previous DriverCommandW.
		push r16

		ldi r16,$02			;chip select - lcd input active. ($02 so the Chip Select
pin is driven low (active), while the RESET pin is kept high (inactive).)
		out PORTc,r16

		ldi r16,$00			;Writes 0b00(0000000) to driver, so it knows the next
byte sent is data to write to a register.
		rcall SPI_MasterTransmit

		mov r16,r17			;move data into r16, so SPI_MasterTransmit method
can send it.
		rcall SPI_MasterTransmit

		ldi r16,$03			;chip select - lcd inactive. ($03 so the Chip Select pin
is driven high (inactive), while the RESET pin is kept high (inactive).)
		out PORTc,r16

		pop r16
		ret

DriverStatusR:				;reads data into r17 from the status register on the driver.
```

```
		push r16

		ldi r16,$02				;chip select - lcd input active. ($02 so the Chip Select
pin is driven low (active), while the RESET pin is kept high (inactive).)
		out PORTC,r16

		ldi r16,$c0				;Writes 0b11(0000000) to driver, so it knows the next
byte sent is data to write to a register.
		rcall SPI_MasterTransmit

		ldi r16,$00				;dummy data transmit, to fill SPDR register with data
from driver
		rcall SPI_MasterTransmit
		in r17, SPDR

		ldi r16,$03				;chip select - lcd inactive. ($03 so the Chip Select pin
is driven high (inactive), while the RESET pin is kept high (inactive).)
		out PORTC,r16

		pop r16
		ret

DriverCommandW:					;write address in r17 to the driver, so subsequent
DriverDataR/DriverDataW calls act on
		push r16

		ldi r16,$02				;chip select - lcd input active. ($02 so the Chip Select
pin is driven low (active), while the RESET pin is kept high (inactive).)
		out PORTC,r16

		ldi r16,$80				;Writes 0b10(0000000) to driver, so it knows the next
byte sent is data to write to a register.
		rcall SPI_MasterTransmit

		mov r16,r17				;Move register address into r16, so
SPI_MasterTransmit method can send it.
		rcall SPI_MasterTransmit

		ldi r16,$03				;chip select - lcd inactive. ($03 so the Chip Select pin
is driven high (inactive), while the RESET pin is kept high (inactive).)
		out PORTC,r16

		pop r16
		ret

WriteReg:					;write data in r18 to driver register with address in
r17
		rcall DriverCommandW	;tell driver we're working with register with address in r17
		mov r17,r18				;move data to write into r17 in preparation for
DriverDataW
		rcall DriverDataW		;write data in r17 to register on driver.
		rcall DriverWait		;wait for driver to no longer be busy.
		ret

ReadReg:					;read data into r17 from driver register with address
in r17
```

```
                rcall DriverCommandW    ;tell driver we're working with register with address in r17
                rcall DriverDataR       ;read data from driver register into r17
                rcall DriverWait        ;wait for driver to no longer be busy.
                Ret

;
; GraphicalMethods.asm
;
; Created: 14/11/2016 15:44:17
; Authors: AL4413, MV914
;

ThreeBytesFromColour:                                   ;creates separate rgb bytes in r22-24 from a
2 byte colour in r25/26
                                                                ;In two bytes,
data stored as RRRRRGGG GGGBBBBB
                ;Extract 5 Red bits from the high byte (r26)
                mov r22, r26    ;RRRRRGGG               ;Move low byte into r22
                LSR r22                 ;0RRRRRGG               ;shift right, to move Red bits into
lowest 5 bits.
                LSR r22                 ;00RRRRRG
                LSR r22                 ;000RRRRR

                ;Extract 6 Green bits from Both bytes.
                mov r23, r25    ;GGGBBBBB               ;Move low byte into r23
                lsr r23                 ;0GGGBBBB               ;Shift right, to move Green bits into
lowest 3 bits.
                lsr r23                 ;00GGGBBB
                lsr r23                 ;000GGGBB
                lsr r23                 ;0000GGGB
                lsr r23                 ;00000GGG
                mov r24,r26             ;RRRRRGGG               ;Move high byte into r24
                ANDI r24,$07    ;00000GGG               ;AND with 0b00000111, to clear Red bits out
of the byte.
                lsl r24                 ;0000GGG0               ;Shift left, to move green bits into
[5:3]
                lsl r24                 ;000GGG00
                lsl r24                 ;00GGG000
                OR r23,r24              ;00GGGGGG               ;OR with r23, to merge both
sections of Green data into the same byte.

                ;Extract 5 Blue bits from the low byte (r25)
                mov r24, r25    ;GGGBBBBB               ;Move high byte into r24
                ANDI r24,$1F    ;000BBBBB               ;AND with 0b00011111, to clear Green bits
out of byte.
                ret

ColourFrom3Bytes:               ;creates a 2 byte colour in r25/26 from the separate rgb bytes in r22-24
                ;r22 = red data          000RRRRR
                ;r23 = green data 00GGGGGG
                ;r24 = blue data   000BBBBB
                push r22
                push r23
                push r24

                ;Create lower byte of 2 byte colour
```

```
                mov r25,r24             ;000BBBBB               ;Move Blue data into r25.
                mov r24,r23             ;00GGGGGG               ;Move Green data into r24, so we
can modify it without losing any data.
                lsl r24                 ;0GGGGGG0               ;Shift Green data left, to get lowest
3 bits into bits [7:5].
                lsl r24                 ;GGGGGG00
                lsl r24                 ;GGGGG000
                lsl r24                 ;GGGG0000
                lsl r24                 ;GGG00000
                OR r25,r24              ;GGGBBBBB               ;merge blue data with lower 3
green bits, to finish r25.
                                                ;r25 done.

                ;Create upper byte of 2 byte colour
                mov r26,r23             ;00GGGGGG               ;Move Green data into r26.
                lsr r26                 ;000GGGGG               ;Shift Green data right, to get top 3
bits into bits [2:0].
                lsr r26                 ;0000GGGG
                lsr r26                 ;00000GGG

                lsl r22                 ;00RRRRR0               ;Shift Red data left, to get it into
bits [7:3]
                lsl r22                 ;0RRRRR00
                lsl r22                 ;RRRRR000
                OR r26,r22              ;RRRRRGGG               ;Merge red data with top 3 green
bits to finish r26.

                pop r24
                pop r23
                pop r22
                ret

SetForegroundColour:                                    ;Set the foreground colour on the driver to
the value stored in r25/26, and update the ForegroundColour in memory with the new value.
                push r17
                push r18
                push r22
                push r23
                push r24
                push r25
                push r26
                push YL
                push YH

                ldi YL, $20                                     ;Load address for the
ForegroundColour into Y register.
                ldi YH, $01

                ST Y+, r25                                      ;Update saved
ForegroundColour with new colour.
                ST Y+, r26

                call ThreeBytesFromColour               ;Break up the 2 Byte colour into three
separate bytes for RGB.
```

```
                ldi r17, $63                    ;Foreground Colour Register 0
(FGCR0). Stores RED bits for foreground colour, bits [4:0].
                mov r18, r22                    ;Red Value
                call WriteReg

                ldi r17, $64                    ;Foreground Colour Register 1
(FGCR1). Stores GREEN bits for foreground colour, bits [5:0].
                mov r18,r23                             ;Green Value
                call WriteReg

                ldi r17, $65                    ;Foreground Colour Register 2
(FGCR2). Stores BLUE bits for foreground colour, bits [4:0].
                mov r18,r24                             ;Blue Value
                call WriteReg

                pop YH
                pop YL
                pop r26
                pop r25
                pop r24
                pop r23
                pop r22
                pop r18
                pop r17
                ret

LoadForegroundColour:                   ;Load ForegroundColour from memory into
r25 and r26
                push ZL
                push ZH

                ldi ZL, $20                     ;Load address for the
ForegroundColour into Z register.
                ldi ZH, $01

                LD r25, Z+                      ;Load ForegroundColour
into registers r25/r26.
                LD r26, Z+

                pop ZH
                pop ZL
                ret

SaveForegroundColourToTemp:             ;Backup current ForegroundColour into
TempForegroundColour in memory.
                push r25
                push r26
                push ZL
                push ZH

                ldi ZL,$20                      ;Load address for the
ForegroundColour into Z register.
                ldi ZH,$01
                ld r25,Z+                       ;Load ForegroundColour into
registers r25/r26.
                ld r26,Z+
```

```
                ST Z+, r25                              ;Save ForegroundColour to
TempForegroundColour in memory.
                ST Z+, r26

                pop ZH
                pop ZL
                pop r26
                pop r25
                ret

LoadForegroundColourFromTemp:           ;Overwrite current ForegroundColour with
TempForegroundColour in memory.
                push r25
                push r26
                push ZL
                push ZH

                ldi ZL,$22                              ;Load address for the
TempForegroundColour into Z register.
                ldi ZH,$01
                ld r25,Z+                       ;Load TempForegroundColour into
registers r25/r26.
                ld r26,Z+
                call SetForegroundColour        ;Set the foreground colour on the driver to the value
stored in r25/26.

                pop ZH
                pop ZL
                pop r26
                pop r25
                ret

Paint:                                          ;Call the appropriate
method according to the value in CurrentDrawingMode.
                push r16
                push r17
                push ZL
                push ZH

                call MouseOnCanvas
                cpi r16, $ff
                BRNE PaintEnd

                ldi ZL,$25                              ;Load address for the
CurrentDrawingMode into Z register.
                ldi ZH,$01
                LD r17,Z                        ;Load CurrentDrawingMode into
r17.

                cpi r17, $00
                in r16,sreg
                SBRC r16,1                              ;If CurrentDrawingMode =
0, call PaintBrush...
                call PaintBrush                 ;...else skip this line.

                cpi r17, $01
```

```
                in r16,sreg
                SBRC r16,1                                      ;If CurrentDrawingMode =
1, call PaintRectangle...
                call PaintRectangle                             ;...else skip this line.

                cpi r17, $02
                in r16,sreg
                SBRC r16,1                                      ;If CurrentDrawingMode =
2, call PaintEllipse...
                call PaintEllipse                       ;...else skip this line.

                cpi r17, $03
                in r16,sreg
                SBRC r16,1                                      ;If CurrentDrawingMode =
3, call PaintTriangle...
                call PaintTriangle                      ;...else skip this line.

                cpi r17, $04
                in r16,sreg
                SBRC r16,1                                      ;If CurrentDrawingMode =
4, call PaintLine...
                call PaintLine                          ;...else skip this line.

                cpi r17, $05
                in r16,sreg
                SBRC r16,1                                      ;If CurrentDrawingMode =
5, call Eraser...
                call Eraser                                     ;...else skip this line.
        PaintEnd:
                pop ZH
                pop ZL
                pop r17
                pop r16
                ret

PaintLine:                                                      ;Takes position clicked on
by user, either saves it, or draws a line between it and the saved position. Immediately repeated points are ignored.
                push r17
                push r19
                push r20
                push r21
                push r22
                push r23
                push r24
                push r27
                push ZL
                push ZH

                ldi ZL,$50                                      ;Load address for the
LineDrawingState into Z register.
                ldi ZH,$01
                LD r17,Z+                                       ;Load LineDrawingState
into r17 to see if this is the first position or the second position for the Line.
                CPI r17,$00                                     ;If LineDrawingState = 0,
then go to PaintLinePoint1...
                BRNE PaintLinePoint2                    ;...else go to PaintLinePoint2
```

```
PaintLinePoint1:                    ;Save point 1 to memory, and change
LineDrawingState to 1.
        call CheckCursorAgainstMemory   ;Compares mouse position to the position stored in
LineDrawingPosition. Returns $ff if they are the same, $00 otherwise.
        cpi r16,$ff
        BREQ PaintLineEnd               ;If saved LineDrawingPosition =
current position, do nothing. I.e. cannot start a line from the same point as last line ended.

        ldi ZL,$50                      ;Load address for the
LineDrawingState into Z register.
        ldi ZH,$01
        ldi r17, $01
        ST Z+,r17                       ;Update LineDrawingState
to 1.

        call CursorLoad                 ;Loads the mouse position into
registers r19-r22
        ST Z+,r19                       ;Save the current mouse
position to the LineDrawingPosition memory addresses.
        ST Z+,r20
        ST Z+,r21
        ST Z+,r22

        rjmp PaintLineEnd               ;Pop back registers from stack, and
return.

PaintLinePoint2:                    ;Read point 1 back from memory, and draw
a Line between it and the current mouse position.
        call CheckCursorAgainstMemory   ;Compares mouse position to the position stored in
LineDrawingPosition. Returns $ff if they are the same, $00 otherwise.
        cpi r16,$ff
        BREQ PaintLineEnd               ;If saved LineDrawingPosition =
current position, do nothing. I.e. cannot have a line starting and ending at the same point.

        ldi ZL,$50                      ;Load address for the
LineDrawingState into Z register.
        ldi ZH,$01
        ldi r17, $00
        ST Z+,r17                       ;Reset status to 0, i.e. no
points specified.

        call CursorLoad                 ;Loads the mouse position into
registers r19-r22
        mov r24, r22                    ;Move current mouse position into
registers r21-r24, in preparation for the DrawLine method.
        mov r23, r21
        mov r22, r20
        mov r21, r19

        LD r17,Z+                       ;Load saved
LineDrawingPosition into r17-r20
        LD r18,Z+
        LD r19,Z+
        LD r20,Z+
```

```
                call DrawLine                                   ;Draws a line between the saved
LineDrawingPosition and the current mouse position.

                ldi ZL,$51                                      ;Load address for the
LineDrawingPosition into Z register.
                ldi ZH,$01
                ST Z+,r21                                       ;Save the current mouse
position to the LineDrawingPosition memory addresses.
                ST Z+,r22
                ST Z+,r23
                ST Z+,r24

        PaintLineEnd:                                           ;pop register values back from stack
and return.
                pop ZH
                pop ZL
                pop r27
                pop r24
                pop r23
                pop r22
                pop r21
                pop r20
                pop r19
                pop r17
                ret

PaintTriangle:                                                  ;Takes position clicked on by user,
either saves it, or draws a triangle with it and the saved positions as vertices. Immediately repeated points are
ignored.
                push r16
                push r17
                push r19
                push r20
                push r21
                push r22
                push r23
                push r24
                push r27
                push r28
                push r29
                push ZL
                push ZH

                ldi ZL,$40                                      ;Load address for the
TriangleDrawingState into Z register.
                ldi ZH,$01
                LD r17,Z                                        ;Load TriangleDrawingState into r17
to see if this is the first, second or third position for the triangle.

                cpi r17,$01                                     ;If TriangleDrawingState =
1, then go to PaintTrianglePoint2...
                BREQ PaintTrianglePoint2

                cpi r17,$02                                     ;...else if
TriangleDrawingState = 2, then go to PaintTrianglePoint3...
                BREQ PaintTrianglePoint3
```

```
PaintTrianglePoint1:                    ;...else go to PaintTrianglePoint1. Save point
1 to memory, and progress TriangleDrawingState to 1
        ldi ZL,$45                      ;Load address for
TriangleDrawingPosition2 into Z register.
        ldi ZH,$01
        call CheckCursorAgainstMemory   ;Compares mouse position to the position stored in
TriangleDrawingPosition2. Returns $ff if they are the same, $00 otherwise.
        cpi r16,$ff
        BREQ PaintTriangleEnd           ;If saved TriangleDrawingPosition2 = current
position, do nothing. I.e. cannot start a triangle from the same point as last triangle ended.

        ldi ZL,$40                      ;Load address for
TriangleDrawingState into Z register.
        ldi ZH,$01
        ldi r17, $01
        ST Z+,r17                       ;Update
TriangleDrawingState to 1.

        call CursorLoad                 ;Loads the mouse position into
registers r19-r22
        ST Z+,r19                       ;Save the current mouse
position to the TriangleDrawingPosition1 memory addresses.
        ST Z+,r20
        ST Z+,r21
        ST Z+,r22

        rjmp PaintTriangleEnd           ;Pop back registers from stack, and return.

PaintTrianglePoint2:                    ;save point 2 to memory, and progress
TriangleDrawingState to 2.
        ldi ZL,$41                      ;Load address for
TriangleDrawingPosition1 into Z register.
        ldi ZH,$01
        call CheckCursorAgainstMemory   ;Compares mouse position to the position stored in
TriangleDrawingPosition1.
        cpi r16,$ff
        BREQ PaintTriangleEnd           ;If saved TriangleDrawingPosition1 = current
position, do nothing. I.e. cannot have a triangle wih identical first two vertices.

        ldi ZL,$40                      ;Load address for
TriangleDrawingState into Z register.
        ldi ZH,$01
        ldi r17, $02                    ;Update TriangleDrawingState to 2.
        ST Z+,r17
        call CursorLoad                 ;Loads the mouse position into
registers r19-r22

        ldi ZL,$45                      ;Load address for
TriangleDrawingPosition2 into Z register.
        ldi ZH,$01
        ST Z+,r19                       ;Save the current mouse
position to the PaintTrianglePoint2 memory addresses
        ST Z+,r20
        ST Z+,r21
```

```
		ST Z+,r22

		rjmp PaintTriangleEnd			;Pop back registers from stack, and return.

	PaintTrianglePoint3:				;Read TriangleDrawingPositions 1 and 2 back
from memory, and draw a triangle using them and the current mouse position as vertices.
		ldi ZL,$45				;Load address for
TriangleDrawingPosition2 into Z register.
		ldi ZH,$01
		call CheckCursorAgainstMemory	;Compares mouse position to the position stored in
TriangleDrawingPosition2.
		cpi r16,$ff
		BREQ PaintTriangleEnd			;If saved TriangleDrawingPosition2 = current
position, do nothing. I.e. cannot have a triangle wih identical last two vertices.

		ldi ZL,$40				;Load address for
TriangleDrawingState into Z register.
		ldi ZH,$01
		ldi r17, $00				;Reset TriangleDrawingState to 0.
		ST Z+,r17

		call CursorLoad				;Loads the mouse position into
registers r19-r22
		mov r28, r22				;Move mouse position into
registers 25-28, in preparation for the DrawTriangle method.
		mov r27, r21
		mov r26, r20
		mov r25, r19

		LD r17,Z+				;Load
TriangleDrawingPosition1 into r17-r20
		LD r18,Z+
		LD r19,Z+
		LD r20,Z+

		LD r21,Z+				;Load
TriangleDrawingPosition1 into r21-r24
		LD r22,Z+
		LD r23,Z+
		LD r24,Z+

		ldi r29,$ff				;Set filled boolean to true.

		call DrawTriangle			;Draw a Triangle with points specified by
r17-r28, and filled boolean flag in r29.

		ldi ZL,$45				;Load address for
TriangleDrawingPosition2 into Z register.
		ldi ZH,$01
		ST Z+,r25				;Save the current mouse
position to the PaintTrianglePoint2 memory addresses.
		ST Z+,r26
		ST Z+,r27
		ST Z+,r28

	PaintTriangleEnd:				;Pop back registers from stack, and return.
```

```
		pop ZH
		pop ZL
		pop r29
		pop r28
		pop r27
		pop r24
		pop r23
		pop r22
		pop r21
		pop r20
		pop r19
		pop r17
		pop r16
		ret

PaintEllipse:					;Takes position clicked on by user,
and either saves it, or draws an ellipse with it and the saved position as a center, and corner of the bounding
rectangle. Immediately repeated points are ignored.
		push r17
		push r19
		push r20
		push r21
		push r22
		push r23
		push r24
		push r25
		push r26
		push r27
		push ZL
		push ZH

		ldi ZL,$35				;Load address for the
EllipseDrawingState into Z register.
		ldi ZH,$01
		LD r17,Z+				;Load EllipseDrawingState
into r17 to see if this is the first or second position for the ellipse.

		cpi r17,$00
		BRNE PaintEllipsePoint2			;If EllipseDrawingState = 0 then Point 1, else
Point 2.

	PaintEllipsePoint1:				;Save point 1 to memory, and
change EllipseDrawingState to 1.
		call CheckCursorAgainstMemory		;Compares mouse position to the position stored in
EllipseDrawingPosition.
		cpi r16,$ff
		BREQ PaintEllipseEnd			;If saved EllipseDrawingPosition = current
position, do nothing. I.e. cannot start an ellipse from the end position of the previous ellipse.

		ldi ZL,$35				;Load address for the
EllipseDrawingState into Z register.
		ldi ZH,$01
		ldi r17, $01
		ST Z+,r17				;Progress
EllipseDrawingState to 1, i.e. origin saved.
```

```
                call CursorLoad                                 ;Loads the current mouse position
into registers r19-r22

                ST Z+,r19                                       ;Save the current mouse
position to the EllipseDrawingPosition memory addresses.
                ST Z+,r20
                ST Z+,r21
                ST Z+,r22

                jmp PaintEllipseEnd                             ;pop back registers, and return.

        PaintEllipsePoint2:                                     ;Read EllipseDrawingPosition back
from memory, and draw an ellipse using it as the centre, and the current mouse position as a bounding rectangle.
                call CheckCursorAgainstMemory           ;Compares mouse position to the position stored in
EllipseDrawingPosition.
                cpi r16,$ff
                BREQ PaintEllipseEnd                    ;if saved position = current position, do
nothing, i.e. cannot have an ellipse with zero size.

                ldi ZL,$35                                      ;Load address for the
EllipseDrawingState into Z register.
                ldi ZH,$01
                ldi r17, $00                                    ;reset status to no points specified.
                ST Z+,r17

                call CursorLoad                                 ;Loads the current mouse position
into registers r19-r22

                mov r26, r22                                    ;move current mouse position into
registers 21-24, in preparation for the DrawEllipse method.
                mov r25, r21
                mov r24, r20
                mov r23, r19


                LD r19,Z+                                       ;Load the saved position
from the EllipseDrawingPosition memory addresses into r19-r22.
                LD r20,Z+
                LD r21,Z+
                LD r22,Z+

                ldi r27,$ff                                     ;Set boolean filled
argument to true.

                call SetActiveWindowToCanvas            ;Set active window to Canvas, so drawing the ellipse
does not overwrite parts of the UI.
                call DrawEllipse                        ;Draw ellipse centred on saved position, and
with x and y axes defined by  difference between saved point x,y positions, and current mouse x,y positions.
                call SetActiveWindowToScreen            ;Reset active window to whole screen, so UI
elements can be redrawn.

                ldi ZL,$36                                      ;Load address for the
EllipseDrawingPosition into Z register.
                ldi ZH,$01
                ldi ZH,$01
```

```
		ST Z+,r23				;Save the current mouse position to the EllipseDrawingPosition memory addresses.
		ST Z+,r24
		ST Z+,r25
		ST Z+,r26


	PaintEllipseEnd:			;pop back registers and return.
		pop ZH
		pop ZL
		pop r27
		pop r26
		pop r25
		pop r24
		pop r23
		pop r22
		pop r21
		pop r20
		pop r19
		pop r17
		ret

PaintRectangle:				;Takes position clicked on by user, and either saves it, or draws a rectangle with it and the saved position as opposite corners of the rectangle. Immediately repeated points are ignored.
		push r17
		push r19
		push r20
		push r21
		push r22
		push r23
		push r24
		push r27
		push ZL
		push ZH

		ldi ZL,$30				;Load address for the RectangleDrawingState into Z register.
		ldi ZH,$01
		LD r17,Z+				;Load RectangleDrawingState into r17 to see if this is the first or second position for the rectangle.

		cpi r17,$00
		BRNE PaintRectanglePoint2		;If RectangleDrawingState = 0 then Point 1, else Point 2.

	PaintRectanglePoint1:			;Save point 1 to memory, and change RectangleDrawingState to 1.
		call CheckCursorAgainstMemory	;Compares mouse position to the position stored in RectangleDrawingPosition.
		cpi r16,$ff
		BREQ PaintRectangleEnd		;If saved RectangleDrawingPosition = current position, do nothing. I.e. cannot start a rectangle from the end position of the previous rectangle.

		ldi ZL,$30				;Load address for the RectangleDrawingState into Z register.
```

```
                ldi ZH,$01
                ldi r17, $01
                ST Z+,r17                       ;Progress
RectangleDrawingState to 1, i.e. one corner already saved.

                call CursorLoad                 ;Loads the mouse position into
registers r19-r22

                ST Z+,r19                       ;Save the current mouse
position to the RectangleDrawingPosition memory addresses.
                ST Z+,r20
                ST Z+,r21
                ST Z+,r22

                rjmp PaintRectangleEnd          ;pop back registers and return.

        PaintRectanglePoint2:                   ;Read RectangleDrawingPosition back from
memory, and draw a rectangle using it and the current mouse position as opposite corners.
                call CheckCursorAgainstMemory   ;Compares mouse position to the position stored in
RectangleDrawingPosition.
                cpi r16,$ff
                BREQ PaintRectangleEnd          ;If saved position = current position, do
nothing, i.e. cannot have a rectangle with identical start and end points.

                ldi ZL,$30                      ;Load address for the
RectangleDrawingState into Z register.
                ldi ZH,$01
                ldi r17, $00
                ST Z+,r17                       ;Reset status to no points
specified.

                call CursorLoad                 ;Loads the current mouse position
into registers r19-r22

                mov r24, r22                    ;Move current mouse position into
registers 21-24, in preparation for the DrawRectangle method.
                mov r23, r21
                mov r22, r20
                mov r21, r19

                LD r17,Z+                       ;Load the saved position
from the RectangleDrawingPosition memory addresses into r19-r22.
                LD r18,Z+
                LD r19,Z+
                LD r20,Z+

                ldi r27,$ff                     ;Set boolean filled
argument to true.

                call DrawRectangle              ;Draw a rectangle of the foreground
colour with corners at the saved position and the current mouse position.

                ldi ZL,$31                      ;Load address for the
RectangleDrawingPosition into Z register.
                ldi ZH,$01
```

```
                ST Z+,r21                                   ;Save the current mouse
position to the RectangleDrawingPosition memory addresses.
                ST Z+,r22
                ST Z+,r23
                ST Z+,r24

        PaintRectangleEnd:                                  ;pop back registers and return.
                pop ZH
                pop ZL
                pop r27
                pop r24
                pop r23
                pop r22
                pop r21
                pop r20
                pop r19
                pop r17
                ret

Eraser:                                                     ;Draw a white circle of
radius BrushSize to the canvas centred at the current mouse position.
                push r25
                push r26

                rcall SaveForegroundColourToTemp    ;Backup current ForegroundColour into
TempForegroundColour in memory.

                ldi r25, COLOUR_WHITEL
                ldi r26, COLOUR_WHITEH
                call SetForegroundColour            ;Set foreground colour to white.

                call PaintBrush                             ;Draws a white circle to the canvas
at the current mouse position (i.e. erase).

                rcall LoadForegroundColourFromTemp  ;Restore foreground colour to its value at the start of
the method.

                pop r26
                pop r25
                ret

PaintPixel:                                                 ;Set the pixel at
coordinates stored in r19-r22 to the current foreground colour.
                push r17
                push r25
                push r26

                call MemCursorWrite                         ;Move Memory cursor on driver to
the position in r19-r22.

                ldi r17, $02                                ;Write to the 'Memory Read/Write
Command' register on the driver, the register that allows us to access the internal memory of the driver.
                call DriverCommandW                         ;Tell the driver to put subsequent
DriverDataW data into register $02.
```

```
                call LoadForegroundColour               ;Load the current foreground colour from
memory into registers r25/r26.

                MOV r17, r26
                call DriverDataW                        ;Write the high colour byte to the Driver
memory.

                MOV r17, r25
                call DriverDataW                        ;Write the low colour byte to the Driver
memory.

                pop r26
                pop r25
                pop r17

                rcall DriverWait                        ;Wait for the WAIT pin from the Driver to be
cleared before continuing.
                ret

PaintBrush:                                             ;Draw a circle of radius BrushSize of
the foreground colour to the canvas centred at the current mouse position.
                push r19
                push r20
                push r21
                push r22
                push r23
                push r27

                call SetActiveWindowToCanvas            ;Set active window to Canvas, so drawing the circle
does not overwrite parts of the UI.

                call CursorLoad                         ;Loads the mouse position into
registers r19-r22.

                ldi ZL,$55                                      ;Load address for the
BrushSize into Z register.
                ldi ZH,$01
                ld r23,Z                                ;Load BrushSize into r23, as the
radius for the circle.
                ldi r27,$ff                             ;Set boolean filled argument to
true.

                call DrawCircle                         ;Draw a circle with centre in r19-
r22, radius in r23, filled boolean in r27

                call PaintPixel                         ;Paint a single pixel in the middle of
the circle, for a circle of 'zero' radius.

                call SetActiveWindowToScreen            ;Reset active window to whole screen, so UI
elements can be redrawn.

        PaintBrushEnd:                                  ;pop registers back and return.
                pop r27
                pop r23
                pop r22
                pop r21
```

```
                pop r20
                pop r19
                ret

FillScreen:                                             ;Fills the screen with the
foreground colour
                push r17
                push r18
                push r19
                push r20
                push r21
                push r22
                push r23
                push r24
                push r27

                ldi r17, $00                            ;Store (0,0) in registers r17-r20 as
top left corner of rectangle.
                ldi r18, $00
                ldi r19, $00
                ldi r20, $00

                ldi r21, $1f                            ;Store (799,479) in registers r21-r24
as bottom right corner of rectangle.
                ldi r22, $03
                ldi r23, $df
                ldi r24, $01

                ldi r27, $ff                            ;Set boolean filled argument to
true.

                rcall DrawRectangle                     ;Draw rectangle over the entire
screen.

                pop r27
                pop r24
                pop r23
                pop r22
                pop r21
                pop r20
                pop r19
                pop r18
                pop r17
                ret

DrawRectangle:                                          ;Draws Rectangle with opposite
corners stored in r17-r20 and r21-r24. Boolean filled flag in r27.
                ;       r17 = x0low
                ;       r18 = x0high
                ;       r19 = y0low
                ;       r20 = y0high
                ;       r21 = x1low
                ;       r22 = x1high
                ;       r23 = y1low
                ;       r24 = y1high
                ;       r27 = booleanfilled
```

```
                push r17
                push r18

                ;Write x0 position to Driver
                push r18                        ;Save x0high to stack, so we can use
r17 and r18 for the WriteReg method.
                mov r18,r17                     ;Move x0low to r18, in
preparation for the WriteReg method.
                ldi r17,$91                     ;Draw Line/Square
Horizontal Start Address Register bits [7:0] (DLHSR0)
                call WriteReg                   ;Writes the data in r18 to the Driver
register address in r17.
                pop r18                         ;Recover x0high from stack
into r18.
                ldi r17,$92                     ;Draw Line/Square
Horizontal Start Address Register bits [9:8] (DLHSR1)
                call WriteReg                   ;Writes the data in r18 to the Driver
register address in r17.

                ;Write y0 position to Driver
                ldi r17,$93                     ;Draw Line/Square Vertical
Start Address Register bits [7:0] (DLVSR0)
                mov r18,r19
                call WriteReg                   ;Writes the data in r18 to the Driver
register address in r17.
                ldi r17,$94                     ;Draw Line/Square Vertical
Start Address Register bit [8] (DLVSR1)
                mov r18,r20
                call WriteReg                   ;Writes the data in r18 to the Driver
register address in r17.

                ;Write x1 position to Driver
                ldi r17,$95                     ;Draw Line/Square
Horizontal End Address Register bits [7:0] (DLHER0)
                mov r18,r21
                call WriteReg                   ;Writes the data in r18 to the Driver
register address in r17.
                ldi r17,$96                     ;Draw Line/Square
Horizontal End Address Register bits [9:8] (DLHER1)
                mov r18,r22
                call WriteReg                   ;Writes the data in r18 to the Driver
register address in r17.

                ;Write y1 position to Driver
                ldi r17,$97                     ;Draw Line/Square Vertical
End Address Register bits [7:0] (DLVER0)
                mov r18,r23
                call WriteReg                   ;Writes the data in r18 to the Driver
register address in r17.
                ldi r17,$98                     ;Draw Line/Square Vertical
End Address Register bit [8] (DLVER1)
                mov r18,r24
                call WriteReg                   ;Writes the data in r18 to the Driver
register address in r17.

                ;Trigger the drawing on the driver.
```

```
                ldi r17,$90                                     ;Drawing Control Register
(DCR). Triggers the Drawing functions on the driver (bit 7), and sets what shape is to be drawn (bits 6-4 and 0).
                cpi r27,$ff
                BREQ RectangleFilled                    ;If filled boolean = $ff, then draw a filled
rectangle...
                rjmp RectangleEmpty                     ;...else draw an empty rectangle.

        RectangleFilled:
                ldi r18,$B0                                     ;0b10110000 - Start
drawing [7] a rectangle [4], and specify as filled [5].
                call WriteReg                           ;Writes the data in r18 to the Driver
register address in r17.
                rjmp DrawRectEnd

        RectangleEmpty:
                ldi r18,$90                                     ;0b10010000 - Start
drawing [7] a rectangle [4], and specify as unfilled [5].
                call WriteReg                           ;Writes the data in r18 to the Driver
register address in r17.

        DrawRectEnd:                                    ;pop registers back and return.
                pop r18
                pop r17
                ret

DrawCircle:                                             ;Draws Circle with centre
stored in r19-r22 and radius in r23. Boolean filled flag in r27.
                ;       r19 = xlow
                ;       r20 = xhigh
                ;       r21 = ylow
                ;       r22 = yhigh
                ;       r23 = radius
                ;       r27 = booleanfilled
                push r17
                push r18

                ;Write centre x position to Driver
                ldi r17, $99                            ;Draw Circle Center Horizontal
Address Register bits [7:0] (DCHR0)
                mov r18, r19
                call WriteReg                           ;Writes the data in r18 to the Driver
register address in r17.
                ldi r17, $9A                            ;Draw Circle Center Horizontal
Address Register bits [9:8] (DCHR1)
                mov r18, r20
                call WriteReg                           ;Writes the data in r18 to the Driver
register address in r17.

                ;Write centre y position to Driver
                ldi r17, $9B                            ;Draw Circle Center Vertical
Address Register bits [7:0] (DCVR0)
                mov r18, r21
                call WriteReg                           ;Writes the data in r18 to the Driver
register address in r17.
                ldi r17, $9C                            ;Draw Circle Center Vertical
Address Register bit [8] (DCVR0)
```

```
                mov r18, r22
                call WriteReg                                   ;Writes the data in r18 to the Driver
register address in r17.

                ;Write radius to Driver
                ldi r17, $9D                                    ;Draw Circle Radius Register (DCRR)
                mov r18, r23
                call WriteReg                                   ;Writes the data in r18 to the Driver
register address in r17.

                ;Trigger the drawing on the driver
                ldi r17, $90                                    ;Drawing Control Register (DCR).
Triggers the Drawing functions on the driver (bit 7), and sets what shape is to be drawn (bits 6-4 and 0).

                cpi r27,$ff
                BREQ CircleFilled                       ;If filled boolean = $ff, then draw a filled
circle...
                rjmp CircleEmpty                        ;...else draw an unfilled one.

        CircleFilled:
                ldi r18,$60                                     ;0b01100000 - Start
drawing a circle [6], and specify as filled [5].
                call WriteReg                                   ;Writes the data in r18 to the Driver
register address in r17.
                rjmp DrawCircleEnd

        CircleEmpty:
                ldi r18,$40                                     ;0b01000000 - Start
drawing a circle [6], and specify as non-filled [5].
                call WriteReg                                   ;Writes the data in r18 to the Driver
register address in r17.

        DrawCircleEnd:                                          ;pop back registers and return.
                pop r18
                pop r17
                ret

DrawEllipse:                                                    ;Draws ellipse with centre stored in
r19-r22 and a corner of bounding rectangle in r23-r26. Boolean filled flag in r27.
                ;supply origin      r19-r22
                ;supply corner r23-r26
                ;filled r27
                push r16
                push r23
                push r24
                push r25
                push r26
                push r30
                push r31

                ;Need to extract the semi-major and semi-minor axes from the two coordinates.
                ;If origin has larger x than corner, then axis = origin x - corner x, else axis = corner x - origin x.
                ;Therefore check top byte first:

                cp r24,r20
                in r16,SREG
```

```
		SBRC r16,0
		jmp CaseNX			;If corner has x less than
origin, then go to CaseNX.

		cp r20,r24
		in r16,SREG
		SBRC r16,0
		jmp CasePX			;If corner has x greater
than origin, then go to CasePX

		;If most significant bytes are equal, check lower byte:
		cp r23,r19
		in r16,SREG
		SBRC r16,0
		jmp CaseNX			;If corner has x less than
origin, then go to CaseNX
							;...else go to
CasePX

	CasePX:				;Corner x > origin x,
therefore axis = corner x - origin x.
		SUB r23,r19
		SBC r24,r20			;Subtract origin x from
corner x.
		jmp CaseY			;Continue to sorting Y-axis.

	CaseNX:				;Corner x < origin x,
therefore axis = origin x - corner x.
		mov r30,r23			;Move corner x into
r30/r31, so we can move origin x into r23/r24.
		mov r31,r24
		mov r23,r19			;Move origin x into r23/r24
		mov r24,r20

		SUB r23,r30			;Subtract corner x from
origin x.
		SBC r24,r31


	CaseY:				;Now sort Y axis:
		;If origin has larger y than corner, then axis = origin y - corner y, else axis = corner y - origin y.
		;Therefore check top byte first:
		cp r26,r22
		in r16,SREG
		SBRC r16,0
		jmp CaseNY			;If corner has y less than
origin, then go to CaseNY.

		cp r22,r26
		in r16,SREG
		SBRC r16,0
		jmp CasePY			;If corner has y greater
than origin, then go to CasePY.

		;If most significant bytes are equal, check lower byte:
		cp r25,r21
```

```
                in r16,SREG
                SBRC r16,0
                jmp CaseNY                      ;If corner has y less than
origin, then go to CaseNY.
                                                ;...else go to
CasePY

        CasePY:                                 ;Corner y > origin y,
therefore axis = corner y - origin y.
                SUB r25,r21
                SBC r26,r22                     ;Subtract origin y from
corner y.
                jmp DrawEllipseEnd              ;Draw ellipse, then pop registers
back and return.

        CaseNY:                                 ;Corner y < origin y,
therefore axis = origin y - corner y.
                mov r30,r25                     ;Move corner y into
r30/r31, so we can move origin y  into r25/r26.
                mov r31,r26
                mov r25,r21                     ;Move origin y into r23/r24
                mov r26,r22

                SUB r25,r30                     ;Subtract corner y from
origin y.
                SBC r26,r31

        DrawEllipseEnd:                         ;Draw ellipse, then pop registers
back and return.
                call DrawEllipseWithAxes        ;Draw ellipse with center in r19-r22, and axes in r23-
r26. Filled flag in r27.
                pop r31
                pop r30
                pop r26
                pop r25
                pop r24
                pop r23
                pop r16
                ret


DrawEllipseWithAxes:                            ;Draw ellipse with center in r19-r22, and
axes in r23-r26. Filled flag in r27.
;       r19 = xlow
;       r20 = xhigh
;       r21 = ylow
;       r22 = yhigh
;       r23 = xaxislow
;       r24 = xaxishigh
;       r25 = yaxislow
;       r26 = yaxishigh
;       r27 = booleanfilled
                push r17
                push r18

                ;Write centre x position to driver:
```

```
                ldi r17, $A5                    ;Draw Ellipse Center Horizontal
Address Register bits [7:0] (DEHR0)
                mov r18, r19
                call WriteReg                   ;Writes the data in r18 to the Driver
register address in r17.
                ldi r17, $A6                    ;Draw Ellipse Center Horizontal
Address Register bits [9:8] (DEHR1)
                mov r18, r20
                call WriteReg                   ;Writes the data in r18 to the Driver
register address in r17.

                ;Write centre y position to driver:
                ldi r17, $A7                    ;Draw Ellipse Center Vertical
Address Register bits [7:0] (DEVR0)
                mov r18, r21
                call WriteReg                   ;Writes the data in r18 to the Driver
register address in r17.
                ldi r17, $A8                    ;Draw Ellipse Center Vertical
Address Register bit [8] (DEVR1)
                mov r18, r22
                call WriteReg                   ;Writes the data in r18 to the Driver
register address in r17.

                ;Write x axis length to driver:
                ldi r17, $A1                    ;Draw Ellipse X-axis Setting Register
bits [7:0] (ELL_A0)
                mov r18, r23
                call WriteReg                   ;Writes the data in r18 to the Driver
register address in r17.
                ldi r17, $A2                    ;Draw Ellipse X-axis Setting Register
bits [9:8] (ELL_A1)
                mov r18, r24
                call WriteReg                   ;Writes the data in r18 to the Driver
register address in r17.

                ;Write y axis length to driver:
                ldi r17, $A3                    ;Draw Ellipse Y-axis Setting Register
bits [7:0] (ELL_B0)
                mov r18, r25
                call WriteReg                   ;Writes the data in r18 to the Driver
register address in r17.
                ldi r17, $A4                    ;Draw Ellipse Y-axis Setting Register
bits [9:8] (ELL_B1)
                mov r18, r26
                call WriteReg                   ;Writes the data in r18 to the Driver
register address in r17.

                ;Trigger the drawing on the driver
                ldi r17, $A0                    ;Draw Ellipse/Ellipse Curve Control
Register. Triggers the Drawing functions on the driver (bit 7), and sets what shape is to be drawn (bits 6-4 and 0).
                cpi r27,$ff
                BREQ DEWAFilled                 ;If boolean filled value =
$ff then draw a filled ellipse...
                rjmp DEWAEmpty                  ;...else draw an empty
ellipse.
```

```
        DEWAFilled:
                ldi r18,$C0                     ;0b11000000 - Start
drawing [7] an ellipse [5:4], and specify as filled [6].
                call WriteReg                   ;Writes the data in r18 to the Driver
register address in r17.
                rjmp DEWAEnd                    ;pop registers back, and return.

        DEWAEmpty:
                ldi r18,$80                     ;0b10000000 - Start
drawing [7] an ellipse [5:4], and specify as unfilled [6].
                call WriteReg                   ;Writes the data in r18 to the Driver
register address in r17.

        DEWAEnd:                                ;Pop registers back, and
return.
                pop r18
                pop r17
                ret

DrawLine:                                       ;Draws a straight line
between point r17-r20 and point r21-r24.
        ;       r17 = x0low
        ;       r18 = x0high
        ;       r19 = y0low
        ;       r20 = y0high
        ;       r21 = x1low
        ;       r22 = x1high
        ;       r23 = y1low
        ;       r24 = y1high

                push r17
                push r18

                ;Write x0 position to Driver:
                push r18                        ;Save x0high to stack, so we can use
r17 and r18 for the WriteReg method.
                mov r18,r17                     ;Move x0low to r18, in
preparation for the WriteReg method.
                ldi r17,$91                     ;Draw Line/Square
Horizontal Start Address Register bits [7:0] (DLHSR0)
                call WriteReg                   ;Writes the data in r18 to the Driver
register address in r17.
                pop r18                         ;Recover x0high from stack
into r18.
                ldi r17,$92                     ;Draw Line/Square
Horizontal Start Address Register bits [9:8] (DLHSR1)
                call WriteReg                   ;Writes the data in r18 to the Driver
register address in r17.

                ;Write y0 position to Driver:
                ldi r17,$93                     ;Draw Line/Square Vertical
Start Address Register bits [7:0] (DLVSR0)
                mov r18,r19
                call WriteReg                   ;Writes the data in r18 to the Driver
register address in r17.
```

```
		ldi r17,$94				;Draw Line/Square Vertical
Start Address Register bit [8] (DLVSR1)
		mov r18,r20
		call WriteReg			;Writes the data in r18 to the Driver
register address in r17.

		;Write x1 position to Driver:
		ldi r17,$95				;Draw Line/Square
Horizontal End Address Register bits [7:0] (DLHER0)
		mov r18,r21
		call WriteReg			;Writes the data in r18 to the Driver
register address in r17.
		ldi r17,$96				;Draw Line/Square
Horizontal End Address Register bits [9:8] (DLHER1)
		mov r18,r22
		call WriteReg			;Writes the data in r18 to the Driver
register address in r17.

		;Write y1 position to Driver:
		ldi r17,$97				;Draw Line/Square Vertical
End Address Register bits [7:0] (DLVER0)
		mov r18,r23
		call WriteReg			;Writes the data in r18 to the Driver
register address in r17.
		ldi r17,$98				;Draw Line/Square Vertical
End Address Register bit [8] (DLVER1)
		mov r18,r24
		call WriteReg			;Writes the data in r18 to the Driver
register address in r17.

		;Trigger the drawing on the driver:
		ldi r17,$90				;Drawing Control Register
(DCR). Triggers the Drawing functions on the driver (bit 7), and sets what shape is to be drawn (bits 6-4 and 0).
		ldi r18,$80				;0b10010000 - Start
drawing [7] a line [4], and specify as filled [5].
		call WriteReg			;Writes the data in r18 to the Driver
register address in r17.

		pop r18					;pop registers back and
return.
		pop r17
		ret


DrawTriangle:					;Draws a triangle specified by
points r17-r20,r21-r24,r25-r28. Boolean filled flag in r29.
		;	r17 = x0low
		;	r18 = x0high
		;	r19 = y0low
		;	r20 = y0high
		;	r21 = x1low
		;	r22 = x1high
		;	r23 = y1low
		;	r24 = y1high
		;	r25 = x2low
		;	r26 = x2high
```

```
;           r27 = y2low
;           r28 = y2high
;           r29 = booleanfilled

            push r17
            push r18

            ;Write x0 position to Driver
            push r18                    ;Save x0high to stack, so we can use
r17 and r18 for the WriteReg method.
            mov r18,r17                 ;Move x0low to r18, in
preparation for the WriteReg method.
            ldi r17,$91                 ;Draw Line/Square
Horizontal Start Address Register bits [7:0] (DLHSR0)
            call WriteReg               ;Writes the data in r18 to the Driver
register address in r17.
            pop r18                     ;Recover x0high from stack
into r18.
            ldi r17,$92                 ;Draw Line/Square
Horizontal Start Address Register bits [9:8] (DLHSR1)
            call WriteReg               ;Writes the data in r18 to the Driver
register address in r17.

            ;Write y0 position to Driver
            ldi r17,$93                 ;Draw Line/Square Vertical
Start Address Register bits [7:0] (DLVSR0)
            mov r18,r19
            call WriteReg               ;Writes the data in r18 to the Driver
register address in r17.
            ldi r17,$94                 ;Draw Line/Square Vertical
Start Address Register bit [8] (DLVSR1)
            mov r18,r20
            call WriteReg               ;Writes the data in r18 to the Driver
register address in r17.

            ;Write x1 position to Driver
            ldi r17,$95                 ;Draw Line/Square
Horizontal End Address Register bits [7:0] (DLHER0)
            mov r18,r21
            call WriteReg               ;Writes the data in r18 to the Driver
register address in r17.
            ldi r17,$96                 ;Draw Line/Square
Horizontal End Address Register bits [9:8] (DLHER1)
            mov r18,r22
            call WriteReg               ;Writes the data in r18 to the Driver
register address in r17.

            ;Write y1 position to Driver
            ldi r17,$97                 ;Draw Line/Square Vertical
End Address Register bits [7:0] (DLVER0)
            mov r18,r23
            call WriteReg               ;Writes the data in r18 to the Driver
register address in r17.
            ldi r17,$98                 ;Draw Line/Square Vertical
End Address Register bit [8] (DLVER1)
            mov r18,r24
```

```
                call WriteReg                                   ;Writes the data in r18 to the Driver
register address in r17.

                ;Write x2 position to Driver
                ldi r17,$A9
                mov r18,r25
                call WriteReg                                   ;Writes the data in r18 to the Driver
register address in r17.
                ldi r17,$AA
                mov r18,r26
                call WriteReg                                   ;Writes the data in r18 to the Driver
register address in r17.

                ;Write y2 position to Driver
                ldi r17,$AB                                             ;Draw Triangle Point 2
Horizontal Address Register bits [7:0] (DTPH0)
                mov r18,r27
                call WriteReg                                   ;Writes the data in r18 to the Driver
register address in r17.
                ldi r17,$AC                                             ;Draw Triangle Point 2
Horizontal Address Register bits [9:8] (DTPH1)
                mov r18,r28
                call WriteReg                                   ;Writes the data in r18 to the Driver
register address in r17.

                ;Trigger the drawing on the driver.
                ldi r17,$90                                             ;Drawing Control Register
(DCR). Triggers the Drawing functions on the driver (bit 7), and sets what shape is to be drawn (bits 6-4 and 0).
                cpi r29,$ff
                BREQ triFilled                                  ;If filled boolean = $ff, then draw a
filled triangle...
                rjmp triEmpty                                   ;...else draw an empty triangle.

        triFilled:
                ldi r18,$A1                                             ;0b10100001 - Start
drawing [7] a triangle [0], and specify as filled [5].
                call WriteReg                                   ;Writes the data in r18 to the Driver
register address in r17.
                rjmp DrawTriEnd                                 ;pop registers back and return.

        triEmpty:
                ldi r18,$81                                             ;0b10000001 - Start
drawing [7] a triangle [0], and specify as unfilled [5].
                call WriteReg                                   ;Writes the data in r18 to the Driver
register address in r17.

        DrawTriEnd:                                                     ;pop registers back and
return.
                pop r18
                pop r17
                call DriverWait
                ret


;
; LCDInitMethods.asm
```

```
;
; Created: 14/11/2016 15:45:32
; Authors: AL4413, MV914
;

DisplayOn:                                      ;Boolean 'on' argument in r16 ($00=off,$ff=on).
                                                        ;Sets the 'LCD Display Off' bit in the 'Power
and Display Control Register' (PWRR)
                push r17
                push r18

                ldi r17, $01                    ;Power and Display Control Register (PWRR). Turns
LCD screen on/off, Sets mode to normal/sleep, can perform a software reset.
                cpi r16, $ff                    ;If r16 = $ff...
                BREQ DisplayOnTrue              ;...then turn on the display
                rjmp DisplayOnFalse             ;...else turn off the display.

        DisplayOnTrue:

                ldi r18,$80                     ;Sets the 'LCD Display Off' bit to 1, turning the screen
on.

                call WriteReg                   ;writes the data in r18 to the Driver register address in r17.

                rjmp DisplayOnEnd               ;clean up registers and return.

        DisplayOnFalse:
                ldi r18,$00                     ;Sets the 'LCD Display Off' bit to 0, turning the screen
off.
                call WriteReg                   ;writes the data in r18 to the Driver register address in r17.

        DisplayOnEnd:                           ;pops values back into registers and returns.

                pop r18
                pop r17
                ret

GPIOX:                                          ;Boolean 'on' argument in r16 ($00=off,$ff=on)
                push r17
                push r18

                ldi r17,$C7
                cpi r16,$ff
                BREQ GPIOXTrue
                rjmp GPIOXFalse

        GPIOXTrue:
                ldi r18,$01
                call WriteReg                   ;writes the data in r18 to the Driver register address in r17.
                rjmp GPIOXEnd

        GPIOXFalse:
                ldi r18,$00
                call WriteReg                   ;writes the data in r18 to the Driver register address in r17.

        GPIOXEnd:
```

```
                pop r18
                pop r17
                ret

PWM1config:                                     ;PWM1 Control Register (P1CR) - PWM1
Enable/Disable in bit 7.
                                                ;Boolean 'on' argument in r16
($00=off,$ff=on), turning backlight on/off
                                                ;clock value in r17. Frequency = Driver
System Clock / (2^r17)
                push r17
                push r18

                mov r18, r17            ;move data into r18 in preparation for WriteReg
                ANDI r18,$0f            ;clear top four bits to ensure bad data doesn't set other bits.
                ldi r17, $8A            ;PWM1 Control Register (P1CR).

                cpi r16,$ff
                breq PWM1configTrue
                rjmp PWM1configFalse

        PWM1configTrue:                 ;sets bit 7 to true in PWM1 Control Register, turning
backlight on.
                ORI r18,$80             ;set bit 7 true - enable backllight.
                call WriteReg           ;writes the data in r18 to the Driver register address in r17.
                rjmp PWM1configEnd

        PWM1configFalse:                ;sets bit 7 to false in PWM1 Control Register, turning backlight
off.
                ORI r18,$00             ;set bit 7 false - disable backlight.
                call WriteReg           ;writes the data in r18 to the Driver register address in r17.

        PWM1configEnd:                  ;pops values back into registers and returns.
                pop r18
                pop r17
                ret

PWM1out:                                        ;writes the value in r16 to the PWM1 Duty Cycle
Register (P1DCR), setting the back light brightness.
                push r18
                push r17

                mov r18,r16
                ldi r17,$8B             ;PWM1 Duty Cycle Register (P1DCR).
                call WriteReg           ;writes the data in r18 to the Driver register address in r17.

                pop r17
                pop r18
                ret

LCDTurnOn:                                      ;Sets configuration values for LCD screen, including
screen size, and VSync/HSync setting.
                push r17
                push r18

                ldi r17,$88             ;Phase Locked Loop Control Register 1 (PLLC1)
```

```
		ldi r18,$0b
		call WriteReg			;writes the data in r18 to the Driver register address in r17.

		ldi r17,$89				;Phase Locked Loop Control Register 2 (PLLC2)
		ldi r18,$02
		call WriteReg			;writes the data in r18 to the Driver register address in r17.
		call DEL15ms			;RA8875 datasheet says delay of >100us necessary - 15ms
should be enough.

		ldi r17,$10				;System Configuration Register (SYSR). Controls 8/16
bit colour, and 8/16 bit microprocessor interface.
		ldi r18,$0c				;16bit colour, 8 bit microprocessor interface
		call WriteReg			;writes the data in r18 to the Driver register address in r17.

		ldi r17,$04				;Pixel Clock Setting Register (PCLK). Sets the relative
pixel clock to the sytem clock, and whether pixel data is fetched on a rising or falling edge.
		ldi r18,$81				;Data fetched on falling edge. Pixel clock period is
twice system clock (i.e. half the speed).
		call WriteReg			;writes the data in r18 to the Driver register address in r17.

		;Horizontal settings
		ldi r17,$14				;Horizontal Display Width Register (HDWR).
Horizontal display width(pixels) = (HDWR + 1)*8
		ldi r18,$63				;$63 = 99 -> (99 + 1) * 8 = 800 pixels wide
		call WriteReg			;writes the data in r18 to the Driver register address in r17.

		ldi r17,$16				;Horizontal Non-Display Period Register (HNDR)
		ldi r18,$03				;Horizontal Non-Display Period (pixels) = (HNDR +
1)*8
		call WriteReg			;writes the data in r18 to the Driver register address in r17.

		ldi r17,$17				;HSYNC Start Position Register (HSTR).
		ldi r18,$03				;HSYNC Start Position = 32 pixels
		call WriteReg			;writes the data in r18 to the Driver register address in r17.

		ldi r17,$18				;HSYNC Pulse Width Register (HPWR). Sets HSYNC
Polarity and the Pulse width of HSYNC. HSYNC Pulse Width (pixels) = (HPW + 1)x8
		ldi r18,$0B				;HPW = 11 -> pulse width = 96 pixels. Polarity = low
active.
		call WriteReg			;writes the data in r18 to the Driver register address in r17.

		;Vertical settings
		ldi r17,$19				;Vertical Display Height Register bits [7:0] (VDHR0).
Vertical pixels = VDHR + 1.
		ldi r18,$df				;$01DF = 479 -> 480 pixels
		call WriteReg			;writes the data in r18 to the Driver register address in r17.

		ldi r17,$1a				;Vertical Display Height Register bit [8] (VDHR1).
Vertical pixels = VDHR + 1.
		ldi r18,$01				;$01DF = 479 -> 480 pixels
		call WriteReg			;writes the data in r18 to the Driver register address in r17.

		ldi r17,$1b				;Vertical Non-Display Period Register bits [7:0]
(VNDR0)
		ldi r18,$20				;Vertical Non-Display area = 32 lines
		call WriteReg			;writes the data in r18 to the Driver register address in r17.
```

```
		ldi r17,$1c			;Vertical Non-Display Period Register bit [8] (VNDR1)
		ldi r18,$00			;Vertical Non-Display area = 32 lines
		call WriteReg		;writes the data in r18 to the Driver register address in r17.

		ldi r17,$1d			;VSYNC Start Position Register bits [7:0] (VSTR0).
		ldi r18,$16			;VSYNC Start Position = 22 pixels
		call WriteReg		;writes the data in r18 to the Driver register address in r17.

		ldi r17,$1e			;VSYNC Start Position Register bit [8] (VSTR0).
		ldi r18,$00			;VSYNC Start Position = 22 pixels
		call WriteReg		;writes the data in r18 to the Driver register address in r17.

		ldi r17,$1f			;VSYNC Pulse Width Register (VPWR). Sets VSYNC
Polarity and the Pulse width of VSYNC. VSYNC Pulse Width (pixels) = (VPW + 1)x8
		ldi r18,$01			;VPW = 1 -> pulse width = 2 pixels. Polarity = low
active.
		call WriteReg		;writes the data in r18 to the Driver register address in r17.

		rcall SetActiveWindowToScreen	;Set Active Window to rectangle from (0,0) to (799,479) - i.e
make entire screen drawable region.

		pop r18
		pop r17
		ret


SetActiveWindowToScreen:		;Set Active Window to rectangle from (0,0) to (799,479) - i.e make
entire screen drawable region.
		push r17
		push r18

		;X0
		ldi r17,$30			;Horizontal Start Point of Active Window bits [7:0]
(HSAW0)
		ldi r18,0			;0 pixels from left of screen.
		call WriteReg		;writes the data in r18 to the Driver register address in r17.
		ldi r17,$31			;Horizontal Start Point of Active Window bit [8]
(HSAW1)
		ldi r18,0
		call WriteReg		;writes the data in r18 to the Driver register address in r17.

		;Y0
		ldi r17,$32			;Vertical Start Point of Active Window bits [7:0]
(VSAW0)
		ldi r18,0			;0 pixels from top of screen.
		call WriteReg
		ldi r17,$33			;Vertical Start Point of Active Window bit [8] (VSAW1)
		ldi r18,0
		call WriteReg		;writes the data in r18 to the Driver register address in r17.

		;X1
		ldi r17,$34			;Horizontal End Point of Active Window bits [7:0]
(HEAW0)
		ldi r18,$1F			;799 pixels from left of screen
		call WriteReg		;writes the data in r18 to the Driver register address in r17.
```

```
                ldi r17,$35                     ;Horizontal End Point of Active Window bit [8]
(HEAW1)
                ldi r18,$03
                call WriteReg           ;writes the data in r18 to the Driver register address in r17.

                ;Y1
                ldi r17,$36                     ;Vertical End Point of Active Window bits [7:0]
(VEAW0)
                ldi r18,$DF                     ;479 pixels from top of screen
                call WriteReg           ;writes the data in r18 to the Driver register address in r17.
                ldi r17,$37                     ;Vertical End Point of Active Window bit [8] (VEAW1)
                ldi r18,$01
                call WriteReg           ;writes the data in r18 to the Driver register address in r17.

                pop r18
                pop r17
                ret


SetActiveWindowToCanvas:        ;Set Active Window to rectangle from (81,0) to (799,479) - i.e make
only the Canvas the drawable region.
                push r17
                push r18

                ;X0
                ldi r17,$30                     ;Horizontal Start Point of Active Window bits [7:0]
(HSAW0)
                ldi r18,81                      ;81 pixels from left of screen.
                call WriteReg           ;writes the data in r18 to the Driver register address in r17.
                ldi r17,$31                     ;Horizontal Start Point of Active Window bit [8]
(HSAW1)
                ldi r18,0
                call WriteReg           ;writes the data in r18 to the Driver register address in r17.

                ;Y0
                ldi r17,$32                     ;Vertical Start Point of Active Window bits [7:0]
(VSAW0)
                ldi r18,0               ;0 pixels from top of screen.
                call WriteReg
                ldi r17,$33                     ;Vertical Start Point of Active Window bit [8] (VSAW1)
                ldi r18,0
                call WriteReg           ;writes the data in r18 to the Driver register address in r17.

                ;X1
                ldi r17,$34                     ;Horizontal End Point of Active Window bits [7:0]
(HEAW0)
                ldi r18,$1F                     ;799 pixels from left of screen
                call WriteReg           ;writes the data in r18 to the Driver register address in r17.
                ldi r17,$35                     ;Horizontal End Point of Active Window bit [8]
(HEAW1)
                ldi r18,$03
                call WriteReg           ;writes the data in r18 to the Driver register address in r17.

                ;Y1
                ldi r17,$36                     ;Vertical End Point of Active Window bits [7:0]
(VEAW0)
```

```
                ldi r18,$DF                     ;479 pixels from top of screen
                call WriteReg           ;writes the data in r18 to the Driver register address in r17.
                ldi r17,$37                     ;Vertical End Point of Active Window bit [8] (VEAW1)
                ldi r18,$01
                call WriteReg           ;writes the data in r18 to the Driver register address in r17.

                pop r18
                pop r17
                ret



;
; CursorMethods.asm
;
; Created: 17/11/2016 16:42:35
; Authors: AL4413, MV914
;

MemCursorWrite:                                                                 ;Writes
to driver board the memory 46-49, cursor position from r19-r22
                push r17
                push r18
                ldi r17, $46
                mov r18, r19
                call WriteReg
                ldi r17, $47
                mov r18, r20
                call WriteReg
                ldi r17, $48
                mov r18, r21
                call WriteReg
                ldi r17, $49
                mov r18, r22
                call WriteReg
                pop r18
                pop r17
                ret


CursorDisplay:                                                                  ;Send the current
mouse position to the Driver.
                call CursorLoad                                                 ;Read cursor
position from memory into r19, r20, r21, r22
                call CursorWrite                                        ;Write the cursor position
from the registers into the driver
                ret

CursorLoad:                                                                     ;Loads
graphics cursor (mouse) position from microprocessor memory into r19-r22
                push YL
                push YH
                ldi YL,$10                                                      ;Sets
address to save cursor position
                ldi YH,$01
```

```
                LD r19, Y+                                  ;Load X
position into r19,r20 registers
                LD r20, Y+
                LD r21, Y+                                  ;Load Y
position into r21, r22 registers
                LD r22, Y+
                pop YH
                pop YL
                ret

CursorWrite:                                                ;Writes mouse
position (graphic cursor) from r19-r22 to driver board.
                push r17
                push r18
                ldi r17, $80
                mov r18, r19
                call WriteReg
                ldi r17, $81
                mov r18, r20
                call WriteReg
                ldi r17, $82
                mov r18, r21
                call WriteReg
                ldi r17, $83
                mov r18, r22
                call WriteReg
                pop r18
                pop r17
                ret

SetCursorShape:                                             ;Graphic cursor
shape
                push r17
                push r18
                ldi r17, $41                                ;Memory address
                ldi r18, $88                                ;Write cursor
shape active
                call WriteReg
                call CursorShapeMouse                       ;Loads mouse icon data to
memory
                call CursorEnable                           ;Writes graphic cursor to
layer 1 of the LCDScreen
                pop r18
                pop r17
                ret

CursorShapeMouse:                                           ;Loads mouse
icon data to memory
                push r17
                push r18
                push ZL
                push ZH
                ldi r17,$85
                ldi r18,$ff
                call WriteReg
```

```
                ldi ZL, low(MouseCursorData*2)
                ldi ZH, high(MouseCursorData*2)          ;Loads binary string (mouse icon)
into Z register
                ldi r17, $02                                                    ;Write to the
'memory' register on the driver
                call DriverCommandW
                ldi r18, $00                                                    ;Set r18 to 0 for
the loop counter

        CursorShapeMouseLoop:
                lpm r17,Z+
                call DriverDataW
                subi r18, $ff
                cpi r18, $00
                in r17,sreg
                SBRS r17,1
                jmp CursorShapeMouseLoop
                nop
                pop ZH
                pop ZL
                pop r18
                pop r17
                ret

CursorEnable:                                                                   ;Tells the driver to
write the graphic cursor to layer 1
                push r17
                push r18
                ldi r17, $41
                ldi r18, $80                                                    ;Write to layer 1
                call WriteReg
                pop r18
                pop r17
                ret


CursorDisable:                                                                  ;Disables writing
the graphic cursor to layer 1
                push r17
                push r18
                ldi r17, $41
                ldi r18, $00
                call WriteReg
                pop r18
                pop r17
                ret

CheckCursorAgainstMemory:                                               ;Compares current mouse
position to position stored in memory at Z register address.

        ;Returns to r16. $ff if positions match. $00 if positions different.
                push r23
                push r24
                push r25
                push r26
                push ZL
```

```
                push ZH


                call CursorLoad                                 ;Loads the mouse
position into registers r19-r22
                ld r23,Z+
                ld r24,Z+
                ld r25,Z+
                ld r26,Z+

                ldi r16, $ff                                    ;Loads $00 to r16,
positions match

                cp r19,r23
                BRNE CCAMFalse
                cp r20,r24
                BRNE CCAMFalse
                cp r21,r25
                BRNE CCAMFalse
                cp r22,r26
                BRNE CCAMFalse

                rjmp CCAMEnd

        CCAMFalse:
                ldi r16, $00                                    ;Loads $00 to r16,
positions different

        CCAMEnd:
                pop ZH
                pop ZL
                pop r26
                pop r25
                pop r24
                pop r23
                ret

MouseOnToolSquare:                                              ;Compare mouse
position r23-r26, to 20x20 square at position r19-r22.
                push r23
                push r24
                push r25
                push r26

                mov r23,r19
                subi r23,$E0
                mov r24,r20
                mov r25,r21
                subi r25,$E0
                mov r26,r22
                call MouseInsideRectangle                       ;Checks if the mouse is in
the SliderBar

                pop r26
                pop r25
                pop r24
```

```
		pop r23
		ret

MouseOnBar:
	;Compare mouse position r23-r26, to 64x16 SliderBar at position r19-r22.
		push r23
		push r24
		push r25
		push r26
		push YL
		push YH

		mov YL,r19
		mov YH,r20
		adiw Y,1
		mov r19,YL
		mov r20,YH

		mov ZL,r21
		mov ZH,r22

		adiw Y, 63
		adiw Z, 15

		mov r23,YL
		mov r24,YH
		mov r25,ZL
		mov r26,ZH
		call MouseInsideRectangle		;Checks if the mouse is in
the SliderBar

		pop YH
		pop YL
		pop r26
		pop r25
		pop r24
		pop r23
		ret

MouseOnColourSquare:				;Compare mouse position
to 20x20 square at position r19-r22. General method for each of the colour squares.
		push r19
		push r20
		push r21
		push r22
		push r23
		push r24
		push r25
		push r26

		mov r23,r19
		subi r23,$EC
		mov r24,r20
		mov r25,r21
		subi r25,$EC
		mov r26,r22
```

```
                call MouseInsideRectangle                       ;Checks if the mouse is in
the correspondent colour square

                pop r26
                pop r25
                pop r24
                pop r23
                pop r22
                pop r21
                pop r20
                pop r19
                ret

MouseOnScreen:                                                  ;Compare mouse
position to screen size
                push r19
                push r20
                push r21
                push r22
                push r23
                push r24
                push r25
                push r26

                ldi     r19, $00                                ;x0(0)
                ldi     r20, $00
                ldi     r21, $00                                ;y0(0)
                ldi     r22, $00
                ldi r23, $1f                                    ;x1(799)
                ldi r24, $03
                ldi r25, $df                                    ;y1(479)
                ldi r26, $01

                call MouseInsideRectangle                       ;Checks if the mouse is in
the Screen

                pop r26
                pop r25
                pop r24
                pop r23
                pop r22
                pop r21
                pop r20
                pop r19
                ret

MouseOnCanvas:                                                  ;Compare mouse
position to canvas (drawing region).
                push r19
                push r20
                push r21
                push r22
                push r23
                push r24
                push r25
                push r26
```

```
                ldi     r19, 81                                         ;x0
                ldi     r20, $00
                ldi     r21, $00                                ;y0
                ldi     r22, $00
                ldi r23, $1f                                    ;x1
                ldi r24, $03
                ldi r25, $df                                    ;y1
                ldi r26, $01

                Call MouseInsideRectangle                       ;Checks if the mouse is in
the Canvas

                pop r26
                pop r25
                pop r24
                pop r23
                pop r22
                pop r21
                pop r20
                pop r19
                ret

MouseInsideRectangle:                                           ;Compare mouse position
to rectangle defined by registers r19-r26

        ;Set r16 = ff if mouse inside region, 00 if mouse outside region.
                push YL
                push YH
                push ZL
                push ZH
                ldi r16, $aa
                mov YL, r19                                             ;Move x0
into Y
                mov YH, r20
                mov ZL, r21                                             ;Move y0
into Z
                mov ZH, r22
                call CursorLoad                                 ;Load cursor
position into r19-r22

        case1:                                                          ;Mouse x
< x0 -> outside

                        cp YH,r20                                       ;if mouse
x > x0, cannot say the mouse is outside region. Go to case 2.
                        in r17,sreg
                        SBRC r17,0
                        jmp case2

                        cp r20,YH                                       ;if mouse
x < x0, mouse outside region. return 0.
                        in r17,sreg
                        SBRC r17,0
                        jmp MIREnd
```

```
                cp YL,r19                                   ;if mouse
x > x0, cannot say the mouse is outside region. Go to case 2.
                in r17,sreg
                SBRC r17,0
                jmp case2

                cp r19,YL                                   ;if mouse
x < x0, mouse outside region. return 0.
                in r17,sreg
                SBRC r17,0
                jmp MIREnd

        case2:                                              ;mouse y
< y0 -> outside
                cp ZH,r22                                   ;if mouse
y > y0, cannot say the mouse is outside region. Go to case 3.
                in r17,sreg
                SBRC r17,0
                jmp case3

                cp r22,ZH                                   ;if mouse
y < y0, mouse outside region. return 0.
                in r17,sreg
                SBRC r17,0
                jmp MIREnd

                cp ZL,r21                                   ;if mouse
y > y0, cannot say the mouse is outside region. Go to case 3.
                in r17,sreg
                SBRC r17,0
                jmp case3

                cp r21,ZL                                   ;if mouse
y < y0, mouse outside region. return 0.
                in r17,sreg
                SBRC r17,0
                jmp MIREnd

        case3:                                              ;mouse x
> x1 -> outside
                mov YL, r23                                 ;move x1
into Y register
                mov YH, r24

                cp r20,YH                                   ;if mouse
x < x1, cannot say the mouse is outside region. Go to case 4.
                in r17,sreg
                SBRC r17,0
                jmp case4

                cp YH,r20                                   ;if mouse
x > x0, mouse outside region. return 0.
                in r17,sreg
                SBRC r17,0
```

```
                    jmp MIREnd

                    cp r19,YL                                               ;if mouse
x < x0, cannot say the mouse is outside region. Go to case 4.
                    in r17,sreg
                    SBRC r17,0
                    jmp case4

                    cp YL,r19                                               ;if mouse
x > x0, mouse outside region. return 0.
                    in r17,sreg
                    SBRC r17,0
                    jmp MIREnd

        case4:                                                              ;mouse y
> y1 -> outside
                    mov ZL, r25                                             ;move y1
into Z register
                    mov ZH, r26

                    cp r22,ZH                                               ;if mouse
y < y0, cannot say the mouse is outside region. Therefore mouse inside region -> Go to MIRPass
                    in r17,sreg
                    SBRC r17,0
                    jmp MIRPass

                    cp ZH,r22                                               ;if mouse
y > y0, mouse outside region. return 0.
                    in r17,sreg
                    SBRC r17,0
                    jmp MIREnd

                    cp r21,ZL                                               ;if mouse
y < y0, cannot say the mouse is outside region. Therefore mouse inside region -> Go to MIRPass.
                    in r17,sreg
                    SBRC r17,0
                    jmp MIRPass

                    cp ZL,r21                                               ;if mouse
y > y0, mouse outside region. return 0.
                    in r17,sreg
                    SBRC r17,0
                    jmp MIREnd

        MIRPass:
                ldi r16, $FF
        MIREnd:
                pop ZH
                pop ZL
                pop YH
                pop YL
                ret


;
; UIMethods.asm
```

```
;
; Created: 22/11/2016 14:03:46
; Authors: AL4413, MV914
;

 DrawUserInterface:
                push r25
                push r26
                push YL
                push YH
                push ZL
                push ZH

                call SaveForegroundColourToTemp         ;Stores current ForegroundColour into
TempForegroundColour in memory
                call DrawSidebar                        ;Draw Grey rectangle as background to user
interface

        ;RED SLIDER BAR
                ldi r25, COLOUR_REDL
                ldi r26, COLOUR_REDH
                call SetForegroundColour                ;Set Foreground Colour to RED.

                ldi YL,7
                ldi YH,0
                ldi ZL,110
                ldi ZH,0

                call DrawSliderBar
                adiw Z,1
                call DrawRedSliderBar                   ;Draw a slider bar of the foreground colour
at position in Y,Z

        ;GREEN SLIDER BAR
                ldi r25, COLOUR_GREENL
                ldi r26, COLOUR_GREENH
                call SetForegroundColour                ;Set Foreground Colour to GREEN.
                ldi YL,7
                ldi YH,0
                ldi ZL,135
                ldi ZH,0

                call DrawSliderBar
                adiw Z,1
                call DrawGreenSliderBar                 ;Draw a slider bar of the foreground colour
at position in Y,Z

        ;BLUE SLIDER BAR
                ldi r25, COLOUR_BLUEL
                ldi r26, COLOUR_BLUEH
                call SetForegroundColour                ;Set Foreground Colour to BLUE.

                ldi YL,7
                ldi YH,0
                ldi ZL,160
                ldi ZH,0
```

```
                call DrawSliderBar                                  ;Draw a slider bar of the foreground
colour at position in Y(x),Z(y)
                adiw Z,1
                call DrawBlueSliderBar

        ;BRUSH SIZE SLIDER BAR
                ldi r25, COLOUR_WHITEL
                ldi r26, COLOUR_WHITEH
                call SetForegroundColour            ;Set Foreground Colour to WHITE.

                ldi YL,7
                ldi YH,0
                ldi ZL,$75
                ldi ZH,$01
                call DrawSliderBar

                call LoadForegroundColourFromTemp

        ;COLOUR SQUARES
                rcall DrawColours
                rcall DrawSelectedColour
                rcall DrawSelectedTool

        ;TOOL ICONS
                call SaveForegroundColourToTemp     ;Backup current ForegroundColour into
TempForegroundColour in memory.

                rcall DrawRectangleTool                 ;First Tool: Draws the rectangle tool icon on
screen.
                rcall DrawEllipseTool                   ;Second Tool: Draws the Ellipse tool icon on
screen.
                rcall DrawTriangleTool                  ;Third Tool: Draws the triangle tool icon on
screen.
                rcall DrawPaintBrushTool            ;Fourth Tool: Draws the Paint Brush tool icon on
screen.
                rcall DrawLineTool                              ;Fifth Tool: Draws the Line tool icon
on screen.
                rcall DrawEraserTool                    ;Sixth Tool: Draws the Eraser tool icon on
screen.

                call LoadForegroundColourFromTemp   ;Reloads saved foreground colour from
TempForegroundColour in memory.

        ;BRUSH SIZE DISPLAY
                rcall DrawBrushSizeTool                 ;Draws a circle of radius = brush size above
the Brush Size slider bar.

                pop ZH
                pop ZL
                pop YH
                pop YL
                pop r26
                pop r25
                ret
```

```
DrawColours:                                    ;To user interface, uploads the 12
colours squares.
        push r25
        push r26
        push YL
        push YH
        push ZL
        push ZH

        call SaveForegroundColourToTemp

        ldi r25, COLOUR_REDL                    ;Square 1, red colour.
        ldi r26, COLOUR_REDH
        call SetForegroundColour
        ldi YL, 5
        ldi YH, 0
        ldi ZL, 5
        ldi ZH, 0
        call DrawColouredSquare

        ldi r25, COLOUR_GREENL                  ;Square 2, green colour.

        ldi r26, COLOUR_GREENH
        call SetForegroundColour
        ldi YL, 30
        ldi YH, 0
        ldi ZL, 5
        ldi ZH, 0
        call DrawColouredSquare

        ldi r25, COLOUR_BLUEL                   ;Square 3, blue colour.

        ldi r26, COLOUR_BLUEH
        call SetForegroundColour
        ldi YL, 55
        ldi YH, 0
        ldi ZL, 5
        ldi ZH, 0
        call DrawColouredSquare

        ldi r25, COLOUR_CYANL                   ;Square 4, cyan  colour.
        ldi r26, COLOUR_CYANH
        call SetForegroundColour
        ldi YL, 5
        ldi YH, 0
        ldi ZL, 30
        ldi ZH, 0
        call DrawColouredSquare

        ldi r25, COLOUR_MAGENTAL                ;Square 5, magenta colour.
        ldi r26, COLOUR_MAGENTAH
        call SetForegroundColour
        ldi YL, 30
        ldi YH, 0
        ldi ZL, 30
        ldi ZH, 0
```

```
        call DrawColouredSquare

        ldi r25, COLOUR_YELLOWL          ;Square 6, yellow colour.
        ldi r26, COLOUR_YELLOWH
        call SetForegroundColour
        ldi YL, 55
        ldi YH, 0
        ldi ZL, 30
        ldi ZH, 0
        call DrawColouredSquare

        ldi r25, COLOUR_WHITEL           ;Square 7, white colour.
        ldi r26, COLOUR_WHITEH
        call SetForegroundColour
        ldi YL, 5
        ldi YH, 0
        ldi ZL, 55
        ldi ZH, 0
        call DrawColouredSquare

        ldi r25, COLOUR_GREYL            ;Square 8, grey colour.
        ldi r26, COLOUR_GREYH
        call SetForegroundColour
        ldi YL, 30
        ldi YH, 0
        ldi ZL, 55
        ldi ZH, 0
        call DrawColouredSquare

        ldi r25, COLOUR_BLACKL           ;Square 9, black colour.
        ldi r26, COLOUR_BLACKH
        call SetForegroundColour
        ldi YL, 55
        ldi YH, 0
        ldi ZL, 55
        ldi ZH, 0
        call DrawColouredSquare

        ldi ZL,$60                       ;Square 10, Custom colour
1.
        ldi ZH,$01
        ld r25, Z+
        ld r26, Z+
        call SetForegroundColour
        ldi YL, 5
        ldi YH, 0
        ldi ZL, 80
        ldi ZH, 0
        call DrawColouredSquare

        ldi ZL,$62                       ;Square 11, Custom colour
2.
        ldi ZH,$01
        ld r25, Z+
        ld r26, Z+
        call SetForegroundColour
```

```
                ldi YL, 30
                ldi YH, 0
                ldi ZL, 80
                ldi ZH, 0
                call DrawColouredSquare

                ldi ZL,$64                                      ;Square 12, Custom colour
3.
                ldi ZH,$01
                ld r25, Z+
                ld r26, Z+
                call SetForegroundColour
                ldi YL, 55
                ldi YH, 0
                ldi ZL, 80
                ldi ZH, 0
                call DrawColouredSquare

                call LoadForegroundColourFromTemp

                pop ZH
                pop ZL
                pop YH
                pop YL
                pop r26
                pop r25
                ret


DrawRectangleTool:                                              ;First Tool: Draws the rectangle
square tool on screen to select "DrawRectangle"
                push r17
                push r18
                push r19
                push r20
                push r21
                push r22
                push r23
                push r24
                push r25
                push r26
                push r27

                ldi r25, COLOUR_WHITEL                  ;Draws filled white square (background of
the icon)
                ldi r26, COLOUR_WHITEH
                call SetForegroundColour
                ldi r17,5                                       ;= x1low
                ldi r18,0                                       ;= x1high
                ldi r19,185                                             ;= y1low
                ldi r20,0                                       ;= y1high
                ldi r21,37                                              ;= x2low
                ldi r22,0                                       ;= x2high
                ldi r23,217                                             ;= y2low
                ldi r24, 0                                      ;= y2high
                ldi r27, $ff                                    ;= booleanfilled
```

```
                call DrawRectangle

                ldi r25, COLOUR_BLACKL          ;Draws non-filled black rectangle around
icon
                ldi r26, COLOUR_BLACKH
                call SetForegroundColour
                ldi r17,5                               := x1low
                ldi r18,0                               := x1high
                ldi r19,185                                     := y1low
                ldi r20,0                               := y1high
                ldi r21,37                                      := x2low
                ldi r22,0                               := x2high
                ldi r23,217                                     := y2low
                ldi r24, 0                              := y2high
                ldi r27, $00                            := boolean nonfilled
                call DrawRectangle

        ;Draws non-filled black rectangle inside icon.
                ldi r17,10                                      := x1low
                ldi r18,0                               := x1high
                ldi r19,194                                     := y1low
                ldi r20,0                               := y1high
                ldi r21,32                                      := x2low
                ldi r22,0                               := x2high
                ldi r23,208                                     := y2low
                ldi r24, 0                              := y2high
                ldi r27, $00                            := boolean nonfilled
                call DrawRectangle

                pop r27
                pop r26
                pop r25
                pop r24
                pop r23
                pop r22
                pop r21
                pop r20
                pop r19
                pop r18
                pop r17
                ret


DrawEllipseTool:                                ;Second Tool: Draws the Ellipse square tool
on screen to select "DrawEllipse"
                push r17
                push r18
                push r19
                push r20
                push r21
                push r22
                push r23
                push r24
                push r25
                push r26
                push r27
```

```
                ldi r25, COLOUR_WHITEL          ;Draws filled white square (background of
the icon)
                ldi r26, COLOUR_WHITEH
                call SetForegroundColour
                ldi r17,43                              := x1low
                ldi r18,0                       := x1high
                ldi r19,185                             := y1low
                ldi r20,0                       := y1high
                ldi r21,75                              := x2low
                ldi r22,0                       := x2high
                ldi r23,217                             := y2low
                ldi r24, 0                      := y2high
                ldi r27, $ff                    := booleanfilled
                call DrawRectangle

                ldi r25, COLOUR_BLACKL          ;Draws non-filled black rectangle around
icon.
                ldi r26, COLOUR_BLACKH
                call SetForegroundColour
                ldi r17,43                              := x1low
                ldi r18,0                       := x1high
                ldi r19,185                             := y1low
                ldi r20,0                       := y1high
                ldi r21,75                              := x2low
                ldi r22,0                       := x2high
                ldi r23,217                             := y2low
                ldi r24, 0                      := y2high
                ldi r27, $00                    := boolean nonfilled
                call DrawRectangle

        ;Draws non-filled black ellipse inside icon.
                ldi r19,59                              := x1low
                ldi r20,0                       := x1high
                ldi r21, 201                    := y1low
                ldi r22,0                       := y1high
                ldi r23,10                              := x2low
                ldi r24,0                       := x2high
                ldi r25,8                       := y2low
                ldi r26, 0                      := y2high
                ldi r27, $00                    := boolean nonfilled
                call DrawEllipseWithAxes

                pop r27
                pop r26
                pop r25
                pop r24
                pop r23
                pop r22
                pop r21
                pop r20
                pop r19
                pop r18
                pop r17
                ret
```

```
DrawTriangleTool:                                   ;Third Tool: Draws the triangle
square tool on screen to select "DrawTriangle"
                push r17
                push r18
                push r19
                push r20
                push r21
                push r22
                push r23
                push r24
                push r25
                push r26
                push r27
                push r28
                push r29

                ldi r25, COLOUR_WHITEL              ;Draws filled white square (background of
the icon)
                ldi r26, COLOUR_WHITEH
                call SetForegroundColour
                ldi r17,5                                   ;= x1low
                ldi r18,0                                   ;= x1high
                ldi r19,222                                 ;= y1low
                ldi r20,0                                   ;= y1high
                ldi r21,37                                          ;= x2low
                ldi r22,0                                   ;= x2high
                ldi r23,254                                 ;= y2low
                ldi r24, 0                                          ;= y2high
                ldi r27, $ff                                ;= booleanfilled
                call DrawRectangle

                ldi r25, COLOUR_BLACKL
                ldi r26, COLOUR_BLACKH
                call SetForegroundColour
                ldi r17,5                                   ;= x1low
                ldi r18,0                                   ;= x1high
                ldi r19,222                                 ;= y1low
                ldi r20,0                                   ;= y1high
                ldi r21,37                                          ;= x2low
                ldi r22,0                                   ;= x2high
                ldi r23,254                                 ;= y2low
                ldi r24, 0                                          ;= y2high
                ldi r27, $00                                ;= booleanfilled
                call DrawRectangle

                                                                    ;Draws non-filled
black triangle inside icon.
                ldi r17,10                                          ;= x1low
                ldi r18,0                                   ;= x1high
                ldi r19,231                                 ;= y1low
                ldi r20,0                                   ;= y1high
                ldi r21,10                                          ;= x2low
                ldi r22,0                                   ;= x2high
                ldi r23,245                                 ;= y2low
                ldi r24, 0                                          ;= y2high
```

```
                ldi r25,32                              ;= x3low
                ldi r26,0                       ;= x3high
                ldi r27,245                     ;= y3low
                ldi r28, 0                              ;= yhigh
                ldi r29, $00                    ;= booleanfilled
                call DrawTriangle

                pop r29
                pop r28
                pop r27
                pop r26
                pop r25
                pop r24
                pop r23
                pop r22
                pop r21
                pop r20
                pop r19
                pop r18
                pop r17
                ret

DrawPaintBrushTool:                                     ;Fourth Tool: Draws the
Pencil square tool on screen to select "Draw"

                push r17
                push r18
                push r19
                push r20
                push r21
                push r22
                push r23
                push r24
                push r25
                push r26
                push r27
                push r28
                push r29

                ldi r25, COLOUR_WHITEL          ;Draws filled white square (background of
the icon)
                ldi r26, COLOUR_WHITEH
                call SetForegroundColour
                ldi r17,43                              ;= x1low
                ldi r18,0                       ;= x1high
                ldi r19,222                             ;= y1low
                ldi r20,0                       ;= y1high
                ldi r21,75                              ;= x2low
                ldi r22,0                       ;= x2high
                ldi r23,254                             ;= y2low
                ldi r24, 0                      ;= y2high
                ldi r27, $ff                    ;= booleanfilled
                call DrawRectangle

                ldi r25, COLOUR_BLACKL          ;Draws non-filled black rectangle around
icon.
```

```
                ldi r26, COLOUR_BLACKH
                call SetForegroundColour
                ldi r17,43                      ;= x1low
                ldi r18,0                       ;= x1high
                ldi r19,222                     ;= y1low
                ldi r20,0                       ;= y1high
                ldi r21,75                      ;= x2low
                ldi r22,0                       ;= x2high
                ldi r23,254                     ;= y2low
                ldi r24, 0                      ;= y2high
                ldi r27, $00                    ;= boolean nonfilled
                call DrawRectangle

                                                ;Draws filled black
triangle inside icon as the pencil peak.
                ldi r17,48                      ;= x0low
                ldi r18,0                       ;= x0high
                ldi r19,227                     ;= y0low
                ldi r20,0                       ;= y0high
                ldi r21,50                      ;= x1low
                ldi r22,0                       ;= x1high
                ldi r23,233                     ;= y1low
                ldi r24, 0                      ;= y1high
                ldi r25,54                      ;= x2low
                ldi r26,0                       ;= x2high
                ldi r27,229                     ;= y2low
                ldi r28, 0                      ;= y2high
                ldi r29, $ff                    ;= boolean filled
                call DrawTriangle

                                                ;Draws first black
line to form pencil icon.
                ldi r17,54                      ;= x0low
                ldi r18,0                       ;= x0high
                ldi r19,229                     ;= y0low
                ldi r20, 0                      ;= y0high
                ldi r21,69                      ;= x1low
                ldi r22,0                       ;= x1high
                ldi r23,244                     ;= y1low
                ldi r24, 0                      ;= y1high
                call DrawLine

                                                ;Draws second
black line to form pencil icon.
                ldi r17,50                      ;= x0low
                ldi r18,0                       ;= x0high
                ldi r19,233                     ;= y0low
                ldi r20, 0                      ;= y0high
                ldi r21,65                      ;= x1low
                ldi r22,0                       ;= x1high
                ldi r23,248                     ;= y1low
                ldi r24, 0                      ;= y1high
                call DrawLine

                                                ;Draws third black
line to form pencil icon.
```

```
                ldi r17,69                              ;= x0low
                ldi r18,0                       ;= x0high
                ldi r19,244                             ;= y0low
                ldi r20, 0                      ;= y0high
                ldi r21,65                              ;= x1low
                ldi r22,0                       ;= x1high
                ldi r23,248                             ;= y1low
                ldi r24, 0                      ;= y1high
                call DrawLine


                pop r29
                pop r28
                pop r27
                pop r26
                pop r25
                pop r24
                pop r23
                pop r22
                pop r21
                pop r20
                pop r19
                pop r18
                pop r17
                ret

DrawLineTool:                                   ;Fifth Tool: Draws the Line square
tool on screen to select "DrawLine"
                push r17
                push r18
                push r19
                push r20
                push r21
                push r22
                push r23
                push r24
                push r25
                push r26
                push r27
                push r28
                push r29

                ldi r25, COLOUR_WHITEL          ;Draws filled white square (background of
the icon)
                ldi r26, COLOUR_WHITEH
                call SetForegroundColour
                ldi r17,5                               ;= x1low
                ldi r18,0                       ;= x1high
                ldi r19,3                       ;= y1low
                ldi r20,1                       ;= y1high
                ldi r21,37                              ;= x2low
                ldi r22,0                       ;= x2high
                ldi r23,35                              ;= y2low
                ldi r24,1                       ;= y2high
                ldi r27, $ff                    ;= boolean filled
                call DrawRectangle
```

```
                ldi r25, COLOUR_BLACKL          ;Draws non-filled black rectangle around
icon.
                ldi r26, COLOUR_BLACKH
                call SetForegroundColour
                ldi r17,5                               ;= x1low
                ldi r18,0                               ;= x1high
                ldi r19,3                               ;= y1low
                ldi r20,1                               ;= y1high
                ldi r21,37                                      ;= x2low
                ldi r22,0                               ;= x2high
                ldi r23,35                                      ;= y2low
                ldi r24,1                               ;= y2high
                ldi r27, $00                            ;= boolean nonfilled
                call DrawRectangle

        ;Draws filled black Line inside icon.
                ldi r17,10                                      ;= x1low
                ldi r18,0                               ;= x1high
                ldi r19,8                               ;= y1low
                ldi r20,1                               ;= y1high
                ldi r21,32                                      ;= x2low
                ldi r22,0                               ;= x2high
                ldi r23,30                                      ;= y2low
                ldi r24,1                               ;= y2high
                ldi r27, $ff                            ;= booleanfilled
                call DrawLine                           ;Draws a line from x0,y0 to x1,y1

                pop r29
                pop r28
                pop r27
                pop r26
                pop r25
                pop r24
                pop r23
                pop r22
                pop r21
                pop r20
                pop r19
                pop r18
                pop r17
                ret

DrawEraserTool:                                 ;Sixth Tool: Draws the Eraser square
tool on screen to select "DrawPencil" in white
                push r17
                push r18
                push r19
                push r20
                push r21
                push r22
                push r23
                push r24
                push r25
                push r26
                push r27
```

```
                push r28
                push r29

                ldi r25, COLOUR_WHITEL          ;Draws filled white square (background of
the icon)
                ldi r26, COLOUR_WHITEH
                call SetForegroundColour
                ldi r17,43                              := x1low
                ldi r18,0                       := x1high
                ldi r19,3                       := y1low
                ldi r20,1                       := y1high
                ldi r21,75                              := x2low
                ldi r22,0                       := x2high
                ldi r23,35                              := y2low
                ldi r24,1                       := y2high
                ldi r27, $ff                    := boolean filled
                call DrawRectangle

                ldi r25, COLOUR_BLACKL          ;Draws non-filled black rectangle around
icon.
                ldi r26, COLOUR_BLACKH
                call SetForegroundColour
                ldi r17,43                              := x1low
                ldi r18,0                       := x1high
                ldi r19,3                       := y1low
                ldi r20,1                       := y1high
                ldi r21,75                              := x2low
                ldi r22,0                       := x2high
                ldi r23,35                              := y2low
                ldi r24,1                       := y2high
                ldi r27, $00                    := booleanfilled
                call DrawRectangle

        ;Draws first black line to form Eraser icon.
                ldi r17,48                              := x1low
                ldi r18,0                       := x1high
                ldi r19,15                              := y1low
                ldi r20,1                       := y1high
                ldi r21,55                              := x2low
                ldi r22,0                       := x2high
                ldi r23,8                       := y2low
                ldi r24,1                       := y2high
                call DrawLine

        ;Draws second black line to form Eraser icon.
                ldi r17,52                              := x1low
                ldi r18,0                       := x1high
                ldi r19,19                              := y1low
                ldi r20,1                       := y1high
                ldi r21,59                              := x2low
                ldi r22,0                       := x2high
                ldi r23,12                              := y2low
                ldi r24,1                       := y2high
                call DrawLine

        ;Draws third black line to form Eraser icon.
```

```
        ldi r17,60                      ;= x1low
        ldi r18,0                   ;= x1high
        ldi r19,27                      ;= y1low
        ldi r20,1                   ;= y1high
        ldi r21,67                      ;= x2low
        ldi r22,0                   ;= x2high
        ldi r23,20                      ;= y2low
        ldi r24,1                   ;= y2high
        call DrawLine

;Draws fourth black line to form Eraser icon.
        ldi r17,55                      ;= x2low
        ldi r18,0                   ;= x2high
        ldi r19,8                   ;= y2low
        ldi r20,1                   ;= y2high
        ldi r21,67                      ;= x2low
        ldi r22,0                   ;= x2high
        ldi r23,20                      ;= y2low
        ldi r24,1                   ;= y2high
        call DrawLine

;Draws fifth black line to form Eraser icon.
        ldi r17,60                      ;= x0low
        ldi r18,0                   ;= x0high
        ldi r19,27                      ;= y0low
        ldi r20,1                   ;= y0high
        ldi r21,48                              ;= x1low
        ldi r22,0                   ;= x1high
        ldi r23,15                      ;= y1low
        ldi r24,1                   ;= y1high
        call DrawLine

;Draws sixth black line to form Eraser icon.
        ldi r17,60                      ;= x1low
        ldi r18,0                   ;= x1high
        ldi r19,27                      ;= y1low
        ldi r20,1                   ;= y1high
        ldi r21,67                      ;= x2low
        ldi r22,0                   ;= x2high
        ldi r23,20                      ;= y2low
        ldi r24,1                   ;= y2high

;Draws  black triangle to fill Eraser icon.
        ldi r25,52                      ;= x1low
        ldi r26,0                   ;= x1high
        ldi r27,19                      ;= y1low
        ldi r28,1                   ;= y1high
        ldi r29, $ff                ;= booleanfilled
        call DrawTriangle

;Draws  black triangle to fill Eraser icon.
        ldi r17,59                      ;= x2low
        ldi r18,0                   ;= x2high
        ldi r19,12                      ;= y2low
        ldi r20,1                   ;= y2high
        ldi r21,67                      ;= x2low
```

```
                ldi r22,0                                       ;= x2high
                ldi r23,20                                              ;= y2low
                ldi r24,1                                       ;= y2high

                ldi r25,52                                              ;= x1low
                ldi r26,0                                       ;= x1high
                ldi r27,19                                              ;= y1low
                ldi r28,1                                       ;= y1high
                ldi r29, $ff                                    ;= booleanfilled
                call DrawTriangle



                pop r29
                pop r28
                pop r27
                pop r26
                pop r25
                pop r24
                pop r23
                pop r22
                pop r21
                pop r20
                pop r19
                pop r18
                pop r17
                ret

DrawBrushSizeTool:                                      ;Seventh Tool-> Draws the brush
square tool on screen
                push r17
                push r18
                push r19
                push r20
                push r21
                push r22
                push r23
                push r24
                push r25
                push r26
                push r27
                push r28
                push r29

                call SaveForegroundColourToTemp

                ldi r25, COLOUR_WHITEL                  ;Draws filled white square (background of
the icon)
                ldi r26, COLOUR_WHITEH
                call SetForegroundColour
                ldi r17, 5                                      ;x1low
                ldi r18, 0                                      ;x1high
                ldi r19, $28                                    ;y1low
                ldi r20, $01                                    ;y1high
                ldi r21, 75                                     ;x2low
                ldi r22, 0                                      ;x2high
```

```
                ldi r23, $6E                                    ;y2low
                ldi r24, $01                                    ;y2high
                ldi r27, $ff                                    ;boolean filled
                call DrawRectangle

                ldi r25, COLOUR_BLACKL                  ;Draws non-filled black rectangle around
icon.
                ldi r26, COLOUR_BLACKH
                call SetForegroundColour
                ldi r27, $00                                    ;boolean non-filled
                call DrawRectangle

                call LoadForegroundColourFromTemp
                ldi ZL,$55                                              ;load BrushSize memory
address into Z register
                ldi ZH,$01
                ld r23,Z                                        ;radiuslow
                ldi r24,0                                       ;radiushigh
                ldi r19,40                                              ;xlow
                ldi r20,0                                       ;xhigh
                ldi r21,$4B                                             ;ylow
                ldi r22,$01                                             ;yhigh
                ldi r27,$ff                                     ;booleanfilled
                call DrawCircle
                call PaintPixel                                                 ;paint a single
pixel in the middle of the circle, for a circle of 'zero' radius.

                ldi r25, COLOUR_BLACKL                  ;Draws non-filled black circle around icon.
                ldi r26, COLOUR_BLACKH
                call SetForegroundColour
                ldi r27,$00                                     ;boolean non-filled
                call DrawCircle


        ;       r19 = xlow
        ;       r20 = xhigh
        ;       r21 = ylow
        ;       r22 = yhigh
        ;       r23 = radiuslow
        ;       r24 = radiushigh
        ;       r27 = booleanfilled

                call LoadForegroundColourFromTemp                       ;Overwrite
current ForegroundColour with TempForegroundColour in memory.

                pop r29
                pop r28
                pop r27
                pop r26
                pop r25
                pop r24
                pop r23
                pop r22
                pop r21
                pop r20
                pop r19
```

```
                pop r18
                pop r17
                ret


DrawGreenSliderBar:                                     ;Draws a slider bar of the
foreground colour at position in Y,Z
                push r16
                push r17
                push r18
                push r19
                push r20
                push r21
                push r22
                push r23
                push r24
                push r25
                push r26
                push YL
                push YH
                push ZL
                push ZH

                ldi r22, 0                                      ;set red value to 0
                ldi r23, $ff                                    ;set green value to -1, so it
is set to 0 for the first loop.
                ldi r24, 0                                      ;set blue value to 0
                ldi r16, $ff

                GreenSliderLoop:                        ;Draws gradient colour of green
inside the green slider bar
                        adiw Y,1                                ;move position on screen
                        subi r23, $ff                           ;increase greenness of
colour
                        subi r16, $ff                           ;increment loop counter
                        push r16
                        push r23                                ;saves Greenness to stack

                        ldi r22, 0                              ;set red value to 0
                        ldi r24, 0                              ;set blue value to 0

                        call ColourFrom3Bytes                   ;creates a 2 byte colour in r25/26
from the separate rgb bytes in r22-24
                        call SetForegroundColour  ;Set the foreground colour on the driver to the value stored in
r25/26.


                        mov r17,YL
                        mov r18,YH
                        mov r19,ZL
                        mov r20,ZH

                        mov r21,YL
                        mov r22,YH
                        adiw Z,13
                        mov r23,ZL
```

```
                mov r24,ZH
                sbiw Z,13
                call DrawLine                   ;Draws a line of the greenness
colour created by colour from 3bytes
                pop r23                         ;loads green back from
stack

                pop r16
                cpi r16,63                      ;checks counter to create
new colour in loop until the end of slider bar (64 pixels long)
                BRNE GreenSliderLoop

        pop ZH
        pop ZL
        pop YH
        pop YL
        pop r26
        pop r25
        pop r24
        pop r23
        pop r22
        pop r21
        pop r20
        pop r19
        pop r18
        pop r17
        pop r16
        ret

DrawBlueSliderBar:                              ;Draws a slider bar of the
foreground colour at position in Y,Z
        push r16
        push r17
        push r18
        push r19
        push r20
        push r21
        push r22
        push r23
        push r24
        push r25
        push r26
        push YL
        push YH
        push ZL
        push ZH

        ldi r22, 0                              ;Set red value to 0
        ldi r23, 0                              ;Set green value to 0
        ldi r24, $ff                            ;Set blue value to -1, so it is set to 0
for the first loop.
        ldi r16, $ff
        BlueSliderLoop:                         ;Draws gradient colour of blue
inside the blue slider bar
                subi r24, $ff                   ;Increase blueness of colour
                subi r16, $ff                   ;Increment loop counter
```

```
                push r16
                push r24                        ;Saves blueness to stack
                ldi r22, 0                      ;Set red value to 0
                ldi r23, 0                      ;Set green value to 0

                call ColourFrom3Bytes
                call SetForegroundColour

                adiw Y,1                        ;Move position on screen
                mov r17,YL
                mov r18,YH
                mov r19,ZL
                mov r20,ZH

                mov r21,YL
                mov r22,YH
                adiw Z,13
                mov r23,ZL
                mov r24,ZH
                sbiw Z,13
                call DrawLine

                adiw Y,1                        ;Move position on screen
                mov r17,YL
                mov r18,YH
                mov r19,ZL
                mov r20,ZH

                mov r21,YL
                mov r22,YH
                adiw Z,13
                mov r23,ZL
                mov r24,ZH
                sbiw Z,13
                call DrawLine                   ;Draws a line of the blueness colour
created by colour from 3bytes

                pop r24                                 ;Loads blue back from
stack

                pop r16
                cpi r16,31
                BRNE BlueSliderLoop             ;Checks counter to create new
colour in loop until the end of slider bar (32  double pixels long)

        pop ZH
        pop ZL
        pop YH
        pop YL
        pop r26
        pop r25
        pop r24
        pop r23
        pop r22
        pop r21
        pop r20
```

```
                pop r19
                pop r18
                pop r17
                pop r16
                ret


DrawRedSliderBar:                                   ;Draws a slider bar of the
foreground colour at position in Y,Z
                push r16
                push r17
                push r18
                push r19
                push r20
                push r21
                push r22
                push r23
                push r24
                push r25
                push r26
                push YL
                push YH
                push ZL
                push ZH

                ldi r22, $ff                        ;Set red value to -1, so it is set to 0
for the first loop.
                ldi r23, 0                          ;Set green value to 0
                ldi r24, 0                          ;Set blue value to 0
                ldi r16, $ff
                RedSliderLoop:                      ;Draws gradient colour of red inside
the red slider bar
                        subi r22, $ff               ;Increase redness of colour
                        subi r16, $ff               ;Increment loop counter
                        push r16
                        push r22                    ;Saves redness to stack
                        ldi r23, 0                  ;Set green value to 0
                        ldi r24, 0                  ;Set blue value to 0

                        call ColourFrom3Bytes
                        call SetForegroundColour

                        adiw Y,1                    ;Move position on screen
                        mov r17,YL
                        mov r18,YH
                        mov r19,ZL
                        mov r20,ZH

                        mov r21,YL
                        mov r22,YH
                        adiw Z,13
                        mov r23,ZL
                        mov r24,ZH
                        sbiw Z,13
                        call DrawLine
```

```
                adiw Y,1                        ;Move position on screen
                mov r17,YL
                mov r18,YH
                mov r19,ZL
                mov r20,ZH

                mov r21,YL
                mov r22,YH
                adiw Z,13
                mov r23,ZL
                mov r24,ZH
                sbiw Z,13
                call DrawLine                   ;Draws a line of the blueness colour
created by colour from 3bytes

                pop r22                             ;Loads red back from stack

                pop r16
                cpi r16,31
                BRNE RedSliderLoop              ;Checks counter to create new
colour in loop until the end of slider bar (32  double pixels long)

        pop ZH
        pop ZL
        pop YH
        pop YL
        pop r26
        pop r25
        pop r24
        pop r23
        pop r22
        pop r21
        pop r20
        pop r19
        pop r18
        pop r17
        pop r16
        ret

DrawSlider:                                     ;Draws a slider at x-
position r21,r22 for a slider bar with top y-position in r23/r24.
        push r17
        push r18
        push r19
        push r20
        push ZL
        push ZH

        mov ZL, r23
        mov ZH, r24
        SBIW Z, 1
        mov r17, r21
        mov r18, r22
        mov r19, ZL                             ;Copies r23 to r19
        mov r20, ZH                             ;Copies r24 to r20
```

```
                call DrawSliderTopPointer ;Draws a small slider triangle with top-corner in r17-r20

                ADIW z, 17                                      ;15 leaves space for 2
                mov r19, ZL
                mov r20, ZH

                call DrawSliderBottomPointer    ;Draws a small slider triangle with bottom-corner in

                pop Zh
                pop ZL
                pop r20
                pop r19
                pop r18
                pop r17
                ret



DrawSliderTopPointer:                                   ;Draws a small slider triangle with bottom-
corner in r17-r20

                push r21
                push r22
                push r23
                push r24
                push r25
                push r26
                push r27
                push r28
                push r29
                push ZL
                push ZH

                call SaveForegroundColourToTemp

        ;x1
                mov ZL, r17                                     ;x0low bottom
                mov ZH, r18                                     ;x0high bottom
                adiw Z,1                                        ;adds one to x0 to
calculate x1
                mov r21, ZL                             ;move x1low to r21
                mov r22, ZH                             ;move x1high to r22
        ;y1
                mov ZL, r19                                     ;y0low bottom
                mov ZH, r20                                     ;y0high bottom
                sbiw Z,1                                ;Substracts one from y0 to calculate
y1
                mov r23, ZL                                     ;move y1low to r23
                mov r24, ZH                                     ;move y1high to r24

        ;x2
                mov ZL, r17                                     ;x0low bottom
                mov ZH, r18                                     ;x0high bottom
                sbiw Z,1                                ;Substracts one from x0 to calculate
x2
```

```
                mov r25, ZL                                     ;Move x2low to r25
                mov r26, ZH                                     ;Move x2high to r26
        ;y2
                mov ZL, r19                                     ;y0low botoom
                mov ZH, r20                                     ;y0high bottom
                sbiw Z,1                                ;Lowers y1  by one and sets as y2
                mov r27, ZL                                     ;Move y2low to r27
                mov r28, ZH                                     ;Move y2high to r28

                ldi r29,$ff                             ;Boolean filled on


                push r17                                ;saves bottom-corner coordinate
                push r18
                push r21
                push r22
                push r27

                ldi r17, 0
                ldi r18, 0
                ldi r21, 80
                ldi r22, 0

                ldi r27,$ff

                ldi r25, COLOUR_GREYL
                ldi r26, COLOUR_GREYH
                call SetForegroundColour

                call DrawRectangle                      ;Draws grey rentangle at same
height of SliderTopPointer to reset previous triangle pointer

                pop r27
                pop r22
                pop r21
                pop r18
                pop r17


                ldi r25, COLOUR_BLACKL
                ldi r26, COLOUR_BLACKH
                call SetForegroundColour                        ;Sets foreground colour
black



                call DrawTriangle                                       ;Draws Triangle at
position stored in r17 to r28
                call LoadForegroundColourFromTemp               ;Loads the colour back to
where it was from temp



                pop ZH
                pop ZL
                pop r29
```

```
                pop r28
                pop r27
                pop r26
                pop r25
                pop r24
                pop r23
                pop r22
                pop r21
                ret


DrawSliderBottomPointer:                        ;Draw a small slider triangle with bottom-
point in r17-r20
                push r21
                push r22
                push r23
                push r24
                push r25
                push r26
                push r27
                push r28
                push r29
                push ZL
                push ZH

                call SaveForegroundColourToTemp

        ;x1
                mov ZL, r17                             ;x0low bottom
                mov ZH, r18                             ;x0high bottom
                adiw Z,1                                ;adds one to x0 to
calculate x1
                mov r21, ZL                             ;move x1low to r21
                mov r22, ZH                             ;move x1high to r22
        ;y1
                mov ZL, r19                             ;y0low bottom
                mov ZH, r20                             ;y0high bottom
                adiw Z,1                                ;adds one to y0 to
calculate y1
                mov r23, ZL                             ;move y1low to
r23
                mov r24, ZH                             ;move y1high to r24

        ;x2
                mov ZL, r17                             ;x0low bottom
                mov ZH, r18                             ;x0high bottom
                sbiw Z,1                                ;Subtracts 1 to x0 to
calculate x2
                mov r25, ZL                             ;move x2low to
r25
                mov r26, ZH                             ;move x2high to r26
        ;y2
                mov ZL, r19                             ;y0low bottom
                mov ZH, r20                             ;y0high bottom
                adiw Z,1                                ;Adds one to y0 to
calculate y2
```

```
                mov r27, ZL                                             ;move y2low to r27
                mov r28, ZH                                             ;move y2high to r28


                push r17
                push r18
                push r21
                push r22
                push r27

                ldi r17, 0
                ldi r18, 0
                ldi r21, 80
                ldi r22, 0

                ldi r25, COLOUR_GREYL
                ldi r26, COLOUR_GREYH
                call SetForegroundColour

                call DrawRectangle                                      ;Draws grey
rentangle at same height of SliderTopPointer to reset previous triangle pointer

                pop r27
                pop r22
                pop r21
                pop r18
                pop r17

                ldi r25, COLOUR_BLACKL
                ldi r26, COLOUR_BLACKH
                call SetForegroundColour                        ;Sets foreground colour black

                ldi r29,$ff                                             ;Boolean
filled on

                call DrawTriangle                                       ;Draw a Triangle with
points specified by r17-r28, and filled boolean flag in r29
                call LoadForegroundColourFromTemp       ;Loads foreground colour back to what stored in
temp

                pop ZH
                pop ZL
                pop r29
                pop r28
                pop r27
                pop r26
                pop r25
                pop r24
                pop r23
                pop r22
                pop r21
                ret


DrawSliderBar:                                                  ;Draws a black slider bar,
at x-position (r17-r20) and y-position (r21-r24)
```

```
                push r17
                push r18
                push r19
                push r20
                push r21
                push r22
                push r23
                push r24
                push r25
                push r26
                push r27
                push YL
                push YH
                push ZL
                push ZH

                mov r17,YL                                      ;move x0low into
r17
                mov r18,YH                                      ;move x0high into
r18
                mov r19,ZL                                      ;move y0low into
r19
                mov r20,ZH                                      ;move y0high into
r20
                adiw Y, 60
                adiw Y, 5                                       ;add 65, length of
the slider bar
                adiw Z, 15
                mov r21, YL                                     ;move x1low into
r21
                mov r22, YH                                     ;move x1high into
r22
                mov r23, ZL                                     ;move y1low into
r23
                mov r24, ZH                                     ;move y1high into
r24
                sbiw Y,60
                sbiw Y,5
                sbiw Z,15
                ldi r27, $ff                                    ;Boolean on
                call DrawRectangle                              ;Draws a non-filled
rectangle using the coordinates stored in r17 to r24

                ldi r25, COLOUR_BLACKL
                ldi r26, COLOUR_BLACKH
                call SetForegroundColour                ;Set the foreground colour on the driver to
black

                mov r17,YL                                      ;move x0low into
r17
                mov r18,YH                                      ;move x0high into
r18
                mov r19,ZL                                      ;move y0low into
r19
                mov r20,ZH                                      ;move y0high into
r20
```

```
                adiw Y, 60
                adiw Y, 5
                adiw Z, 15
                mov r21, YL                                     ;move x1low into
r21
                mov r22, YH                                     ;move x1high into
r22
                mov r23, ZL                                     ;move y1low into
r23
                mov r24, ZH                                     ;move y1high into
r24
                ldi r27, $00                                    ;Boolean off
                call DrawRectangle                              ;Draws a non-filled
rectangle using the coordinates stored in r17 to r24


                pop ZH
                pop ZL
                pop YH
                pop YL
                pop r27
                pop r26
                pop r25
                pop r24
                pop r23
                pop r22
                pop r21
                pop r20
                pop r19
                pop r18
                pop r17
                ret


DrawColouredSquare:                                             ;Draws a 20x20 black
square, at x-position (r17-r20) and y-position (r21-r24)
                push r17
                push r18
                push r19
                push r20
                push r21
                push r22
                push r23
                push r24
                push r25
                push r26
                push r27
                push YL
                push YH
                push ZL
                push ZH

                mov r17, YL                                     ;move x0low into
r17
```

```
                mov r18, YH                                     ;move x0high into
r18
                mov r19, ZL                                     ;move y0low into
r19
                mov r20, ZH                                     ;move y0high into
r20
                adiw Y, 20
                adiw Z, 20
                mov r21, YL                                     ;move x1low into
r21
                mov r22, YH                                     ;move x1high into
r22
                mov r23, ZL                                     ;move y1low into
r23
                mov r24, ZH                                     ;move y1high into
r24
                sbiw Y,20
                sbiw Z,20
                ldi r27, $ff                                    ;Boolean on
                call DrawRectangle                              ;Draws a filled rectangle
using the coordinates stored in r17 to r24

                ldi r25, COLOUR_BLACKL
                ldi r26, COLOUR_BLACKH
                call SetForegroundColour                ;Set the foreground colour on the driver to
black


                mov r17, YL                                     ;move x0low into
r17
                mov r18, YH                                     ;move x0high into
r18
                mov r19, ZL                                     ;move y0low into
r19
                mov r20, ZH                                     ;move y0high into
r20
                adiw Y, 20
                adiw Z, 20
                mov r21, YL                                 ;move x1low into r21
                mov r22, YH                                     ;move x1high into
r22
                mov r23, ZL                                     ;move y1low into
r23
                mov r24, ZH                                     ;move y1high into
r24
                ldi r27, $00                                    ;Boolean off
                call DrawRectangle                              ;Draws a non-filled
rectangle using the coordinates stored in r17 to r24

                pop ZH
                pop ZL
                pop YH
                pop YL
                pop r27
                pop r26
                pop r25
```

```
                pop r24
                pop r23
                pop r22
                pop r21
                pop r20
                pop r19
                pop r18
                pop r17
                ret


DrawSidebar:
                push r17
                push r18
                push r19
                push r20
                push r21
                push r22
                push r23
                push r24
                push r25
                push r26
                push r27
                ldi r25, COLOUR_GREYL
                ldi r26, COLOUR_GREYH
                call SetForegroundColour                ;Set the foreground colour on the driver to
GREY
                ldi r17, $00
                ldi r18, $00
                ldi r19, $00
                ldi r20, $00
                ldi r21, 80
                ldi r22, $00
                ldi r23, $df
                ldi r24, $01
                ldi r27, $ff                                    ;Boolean on
                call DrawRectangle                              ;Draws a non-filled
rectangle using the coordinates stored in r17 to r24
                pop r27
                pop r26
                pop r25
                pop r24
                pop r23
                pop r22
                pop r21
                pop r20
                pop r19
                pop r18
                pop r17
                ret

DrawSelectedColour:                                             ;Draws a selection box
around the ColourSquareSelected square from memory
                push r16
                push r19
                push r20
```

```
                push r21
                push r22
                push ZL
                push ZH
                ldi ZL,$56
                ldi ZH,$01
                ld r16,Z                                ;Loads the
ColourSquareSelected memory value into r16

                ldi r19,5
                ldi r20,0
                ldi r21,5
                ldi r22,0
                cpi r16,1                               ;if colour 1 selected, draw
Selection box around it
                BREQ SelectedColour1
                rcall ColourDeselected                  ;else draw Deselection box around
it (grey box)
                rjmp TestColour2
          SelectedColour1:
                rcall ColourSelected

          TestColour2:
                ldi r19,30
                ldi r20,0
                ldi r21,5
                ldi r22,0
                cpi r16,2                               ;if colour 2 selected, draw
Selection box around it
                BREQ SelectedColour2
                rcall ColourDeselected                  ;else draw Deselection box around
it (grey box)
                rjmp TestColour3
          SelectedColour2:
                rcall ColourSelected

          TestColour3:
                ldi r19,55
                ldi r20,0
                ldi r21,5
                ldi r22,0
                cpi r16,3                               ;if colour 3 selected, draw
Selection box around it
                BREQ SelectedColour3
                rcall ColourDeselected                  ;else draw Deselection box around
it (grey box)
                rjmp TestColour4
          SelectedColour3:
                rcall ColourSelected

          TestColour4:
                ldi r19,5
                ldi r20,0
                ldi r21,30
                ldi r22,0
```

```
                cpi r16,4                                   ;if colour 4 selected, draw
Selection box around it
                BREQ SelectedColour4
                rcall ColourDeselected                      ;else draw Deselection box around
it (grey box)
                rjmp TestColour5
            SelectedColour4:
                rcall ColourSelected

            TestColour5:
                ldi r19,30
                ldi r20,0
                ldi r21,30
                ldi r22,0
                cpi r16,5                                   ;if colour 5 selected, draw
Selection box around it
                BREQ SelectedColour5
                rcall ColourDeselected                      ;else draw Deselection box around
it (grey box)
                rjmp TestColour6
            SelectedColour5:
                rcall ColourSelected

            TestColour6:
                ldi r19,55
                ldi r20,0
                ldi r21,30
                ldi r22,0
                cpi r16,6                                   ;if colour 6 selected, draw
Selection box around it
                BREQ SelectedColour6
                rcall ColourDeselected                      ;else draw Deselection box around
it (grey box)
                rjmp TestColour7
            SelectedColour6:
                rcall ColourSelected

            TestColour7:
                ldi r19,5
                ldi r20,0
                ldi r21,55
                ldi r22,0
                cpi r16,7                                   ;if colour 7 selected, draw
Selection box around it
                BREQ SelectedColour7
                rcall ColourDeselected                      ;else draw Deselection box around
it (grey box)
                rjmp TestColour8
            SelectedColour7:
                rcall ColourSelected

            TestColour8:
                ldi r19,30
                ldi r20,0
                ldi r21,55
                ldi r22,0
```

```
                cpi r16,8                                   ;if colour 8 selected, draw
Selection box around it
                BREQ SelectedColour8
                rcall ColourDeselected                      ;else draw Deselection box around
it (grey box)
                rjmp TestColour9
            SelectedColour8:
                rcall ColourSelected

            TestColour9:
                ldi r19,55
                ldi r20,0
                ldi r21,55
                ldi r22,0
                cpi r16,9                                   ;if colour 9 selected, draw
Selection box around it
                BREQ SelectedColour9
                rcall ColourDeselected                      ;else draw Deselection box around
it (grey box)
                rjmp TestColour10
            SelectedColour9:
                rcall ColourSelected

            TestColour10:
                ldi r19,5
                ldi r20,0
                ldi r21,80
                ldi r22,0
                cpi r16,10                                  ;if colour 10
selected, draw Selection box around it
                BREQ SelectedColour10
                rcall ColourDeselected                      ;else draw Deselection box around
it (grey box)
                rjmp TestColour11
            SelectedColour10:
                rcall ColourSelected

            TestColour11:
                ldi r19,30
                ldi r20,0
                ldi r21,80
                ldi r22,0
                cpi r16,11                                  ;if colour 11
selected, draw Selection box around it
                BREQ SelectedColour11
                rcall ColourDeselected                      ;else draw Deselection box around
it (grey box)
                rjmp TestColour12
            SelectedColour11:
                rcall ColourSelected

            TestColour12:
                ldi r19,55
                ldi r20,0
                ldi r21,80
                ldi r22,0
```

```
                cpi r16,12                                          ;if colour 12
selected, draw Selection box around it
                BREQ SelectedColour12
                rcall ColourDeselected
                rjmp DrawSelectedColourEnd                          ;else draw Deselection box around
it (grey box)
        SelectedColour12:
                rcall ColourSelected

        DrawSelectedColourEnd:
                pop ZH
                pop ZL
                pop r22
                pop r21
                pop r20
                pop r19
                pop r16
                ret

DrawSelectedTool:                                                   ;Draws a selection box
around the ToolSquareSelected square, with coordinates in r19 to r22, from memory.
                push r16
                push r19
                push r20
                push r21
                push r22
                push ZL
                push ZH
                ldi ZL,$25
                ldi ZH,$01
                ld r16,Z                                            ;load the ToolSelected
memory value into r19

                ldi r19,43
                ldi r20,0
                ldi r21,222
                ldi r22,0
                cpi r16,0                                           ;if Tool 1 selected, draw
Selection box around it
                BREQ SelectedTool0
                rcall ToolDeselected                                ;else draw Deselection box around
it
                rjmp TestTool1
        SelectedTool0:
                rcall ToolSelected

        TestTool1:                                                  ;Coordinates of
the box in r19 to r22
                ldi r19,5 ;
                ldi r20,0
                ldi r21,185
                ldi r22,0
                cpi r16,1                                           ;if Tool 2 selected, draw
Selection box around it
                BREQ SelectedTool1
```

```
                rcall ToolDeselected                    ;else draw Deselection box around
it
                rjmp TestTool2
            SelectedTool1:
                rcall ToolSelected

        TestTool2:                                      ;Coordinates of
the box in r19 to r22
                ldi r19,43
                ldi r20,0
                ldi r21,185
                ldi r22,0
                cpi r16,2                               ;if Tool 2 selected, draw
Selection box around it
                BREQ SelectedTool2
                rcall ToolDeselected                    ;else draw Deselection box around
it
                rjmp TestTool3
            SelectedTool2:
                rcall ToolSelected

        TestTool3:                                      ;Coordinates of
the box in r19 to r22
                ldi r19,5
                ldi r20,0
                ldi r21,222
                ldi r22,0
                cpi r16,3                               ;if Tool 3 selected, draw
Selection box around it
                BREQ SelectedTool3
                rcall ToolDeselected                    ;else draw Deselection box around
it
                rjmp TestTool4
            SelectedTool3:
                rcall ToolSelected

        TestTool4:
                ldi r19,5
                ldi r20,0
                ldi r21,3
                ldi r22,1
                cpi r16,4                               ;if Tool 4 selected, draw
Selection box around it
                BREQ SelectedTool4
                rcall ToolDeselected                    ;else draw Deselection box around
it
                rjmp TestTool5
            SelectedTool4:
                rcall ToolSelected


        TestTool5:
                ldi r19,43
                ldi r20,0
                ldi r21,3
                ldi r22,1
```

```
                cpi r16,5                                       ;if Tool 5 selected, draw
Selection box around it
                BREQ SelectedTool5
                rcall ToolDeselected                            ;else draw Deselection box around
it
                rjmp DrawSelectedToolEnd
            SelectedTool5:
                rcall ToolSelected

           DrawSelectedToolEnd:
                pop ZH
                pop ZL
                pop r22
                pop r21
                pop r20
                pop r19
                pop r16
                ret

ColourSelected:                                                 ;take the top left coord of
a colour square in r19 to r22, and draw a BLACK square around it.
                push r25
                push r26

                call SaveForegroundColourToTemp ;Save the foreground colour from driver to memory
                ldi r25, COLOUR_BLACKL
                ldi r26, COLOUR_BLACKH
                call SetForegroundColour                        ;Set the foreground colour on the driver to
BLACK
                rcall ColourSelectionBox                        ;take the top left coord of a colour square in
r19-22, and draw a square of the ForegroundColour around it.

                call LoadForegroundColourFromTemp;Set back the saved foreground colour from memory to
driver

                pop r26
                pop r25
                ret

ColourDeselected:                                               ;take the top left coord of
a colour square in r19-22, and draw a GREY square around it.
                push r25
                push r26

                call SaveForegroundColourToTemp ;Save the foreground colour from driver to memory
                ldi r25, COLOUR_GREYL
                ldi r26, COLOUR_GREYH
                call SetForegroundColour                        ;Set the foreground colour on the driver to
GREY
                rcall ColourSelectionBox                        ;take the top left coord of a colour square in
r19-22, and draw a square of the ForegroundColour around it.

                call LoadForegroundColourFromTemp;Set back the saved foreground colour from memory to
driver

                pop r26
```

```
                pop r25
                ret

ToolSelected:                                                   ;take the top left coord of
a colour square in r19-22, and draw a BLACK square around it.
                push r25
                push r26

                call SaveForegroundColourToTemp ;Save the foreground colour from driver to memory
                ldi r25, COLOUR_BLACKL
                ldi r26, COLOUR_BLACKH
                call SetForegroundColour                        ;Set the foreground colour on the driver to
BLACK
                rcall ToolSelectionBox                          ;take the top left coord of a tool
square in r19-22, and draw a square of the ForegroundColour around it.

                call LoadForegroundColourFromTemp;Set back the saved foreground colour from memory to
driver

                pop r26
                pop r25
                ret

ToolDeselected:                                                 ;take the top left coord of
a colour square in r19-22, and draw a GREY square around it.
                push r25
                push r26

                call SaveForegroundColourToTemp ;Save foreground colour from before and safe to memory
                ldi r25, COLOUR_GREYL
                ldi r26, COLOUR_GREYH
                call SetForegroundColour                        ;Set the foreground colour on the driver to
GREY
                rcall ToolSelectionBox                          ;take the top left coord of a tool
square in r19-22, and draw a square of the ForegroundColour around it.

                call LoadForegroundColourFromTemp;Set back the saved foreground colour from memory to
driver

                pop r26
                pop r25
                ret

ColourSelectionBox:                                             ;take the top left coord of
a colour square in r19-22, and draw a square of the ForegroundColour around it.
                push r16
                push r17
                push r18
                push r19
                push r20
                push r21
                push r22
                push r23
                push r24
                push r25
                push r26
```

```
                mov r17,r19                                     ;move xlow to r17
                mov r18,r20                                     ;move xhigh to r18
                mov r19,r21                                             ;move ylow to r19
                mov r20,r22                                             ;move yhigh to
r20

                                        ;subtract 1 from x and y position, so selection box will be around colour square
                ldi r16,$00
                subi r17,$01                                    ;detract 1 to xlow

                sbc  r18,r16                                    ;leaves xhigh same value
                subi r19,$01                                    ;detract 1 to ylow
                sbc  r20,r16                                    ;leaves yhigh same value

                mov r21,r17                                             ;copies xlow to
r21
                mov r22,r18                                     ;copies xhigh to r22
                mov r23,r19                                     ;copies ylow to r23
                mov r24,r20                                     ;copies yhigh to r24

                ldi r25,22                                              ;Values used for 2
byte addition below
                ldi r26,0

                add r21,r25                                             ;Add 22 to x and y
of point 2, to get other corner of selection box
                adc r22,r26
                add r23,r25
                adc r24,r25


                ldi r27, $00                                    ;Boolean off
                call DrawRectangle                              ;Draws a rectangle at
coordinates in r17 to r24

                pop r26
                pop r25
                pop r24
                pop r23
                pop r22
                pop r21
                pop r20
                pop r19
                pop r18
                pop r17
                pop r16
                ret

ToolSelectionBox:                                               ;take the top left coord of
a tool square in r19-22, and draw a square of the ForegroundColour around it.
                push r16
                push r17
                push r18
                push r19
                push r20
```

```
                push r21
                push r22
                push r23
                push r24
                push r25
                push r26

                mov r17,r19                                     ;move xlow to r17
                mov r18,r20                                     ;move xhigh to
r18
                mov r19,r21                                     ;move ylow to r19
                mov r20,r22                                     ;move yhigh to
r20

                                ;subtract 1 from x and y position, so selection box will be around colour square
                ldi r16,$00
                subi r17,$01                                    ;detract 1 to xlow  to r17

                sbc  r18,r16                                    ;leaves xhigh same value
                subi r19,$01                                    ;detract 1 to ylow to r19
                sbc  r20,r16                                    ;leaves yhigh same value

                mov r21,r17                                     ;copies xlow to
r21
                mov r22,r18                                     ;copies xhigh to
r22
                mov r23,r19                                     ;copies ylow to
r23
                mov r24,r20                                     ;copies yhigh to
r24

                ldi r25,34                                      ;values used for 2
byte addition below.
                ldi r26,0

                add r21,r25                                     ;add 34 to x and y
of point 2, to get other corner of selection box.
                adc r22,r26
                add r23,r25
                adc r24,r25


                ldi r27, $00                                    ;Boolean off
                call DrawRectangle                              ;Draws a rectangle at
coordinates in r17 to r24

                pop r26
                pop r25
                pop r24
                pop r23
                pop r22
                pop r21
                pop r20
                pop r19
                pop r18
                pop r17
```

```
                pop r16
                ret

SelectRedSliderColour:                                  ;Returns the clicked red value in
r22, and updates the selected custom colour with the new red value.
                push r16
                push r17
                push r18
                push r19
                push r20
                push r21

                ldi r19,7                               ;load position of top left
corner of slider into r19 to r22
                ldi r20,0
                ldi r21,110
                ldi r22,0
                call MouseOnBar                         ;returns $ff if cursor on
slider, else $00

                cpi r16, $ff

                BRNE SelectRedSliderColourEnd   ;if r16 not equal to $ff...
                                                                        ;... NOP
                call CursorLoad                         ;else, loads mouse position
into r19-r22
                subi r19,7                                      ;Detracts 7 from
yhigh coordinate
                lsr r19                                         ;Divides it by 2,
since there are 32 out of 64 points available for colour selection
                mov r22,r19                                     ;Copies yhigh
value to r22

                push ZL
                push ZH

                ldi ZL,$57                                      ;update slider
value in memory.
                ldi ZH,$01
                ST Z, r22                               ;Stores yhigh value in Z
register

                pop ZH
                pop ZL

                call UpdateSavedColour

                call CursorLoad
                mov r21,r19
                mov r22,r20
                ldi r23,110
                ldi r24,0

                call DrawSlider                         ;draw a slider at x-position
r21,r22 for a slider bar with top y-position in r23/r24.
```

```
SelectRedSliderColourEnd:                    ;NOP
                pop r21
                pop r20
                pop r19
                pop r18
                pop r17
                pop r16
                ret

SelectGreenSliderColour:                     ;returns the clicked green value in r23, and
updates the selected custom colour with the new green value
                push r16
                push r17
                push r18
                push r19
                push r20
                push r21
                push r22
                                                                        ;load
position of top left corner of slider into r19 to r22
                ldi r19,7
                ldi r20,0
                ldi r21,135
                ldi r22,0
                call MouseOnBar                                 ;returns $ff if cursor on
slider, else $00

                cpi r16, $ff

                BRNE SelectGreenSliderColourEnd  ;if r16 not equal to $ff..
                                                                        ;... NOP
                call CursorLoad                                 ;else, loads mouse position
into r19-r22
                subi r19,7                                              ;Detracts 7 from
yhigh coordinate
                mov r23,r19                                             ;Copies yhigh
value to r22

                push ZL
                push ZH

                ldi ZL,$58                                              ;Updates slider
value in memory.
                ldi ZH,$01
                ST Z, r23                                       ;Stores yhigh value in Z
register

                pop ZH
                pop ZL

                call UpdateSavedColour

                call CursorLoad                                 ;else, loads mouse position
into r19-r22
                mov r21,r19
                mov r22,r20
```

```
                ldi r23,135
                ldi r24,0

                call DrawSlider                                 ;Draws a slider at x-
position r21,r22 for a slider bar with top y-position in r23/r24.

        SelectGreenSliderColourEnd:                             ;NOP
                pop r22
                pop r21
                pop r20
                pop r19
                pop r18
                pop r17
                pop r16
                ret

SelectBlueSliderColour:                                         ;returns the clicked blue value in
r24, and updates the selected custom colour with the new blue value
                push r16
                push r17
                push r18
                push r19
                push r20
                push r21
                push r22

                ldi r19,7                                       ;load position of top left
corner of slider into r19 to r22
                ldi r20,0
                ldi r21,160
                ldi r22,0
                call MouseOnBar                                 ;returns $ff if cursor on
slider, else $00

                cpi r16, $ff
                BRNE SelectBlueSliderColourEnd     ;if r16 not equal to $ff..
                                                                ;... NOP
                call CursorLoad                                 ;else, loads mouse position
into r19-r22
                subi r19,7                                              ;Detracts 7 from
yghigh coordinate
                lsr r19                                                 ;Divides it by 2,
since there are 32 out of 64 points available for colour selection
                mov r24,r19


                push ZL
                push ZH

                ldi ZL,$59                                              ;Updates slider
value in memory
                ldi ZH,$01
                ST Z, r24                                       ;Stores yhigh value in Z
register

                pop ZH
```

```
                pop ZL

                call UpdateSavedColour

                call CursorLoad                                 ;Loads graphics cursor
(mouse) position from microprocessor memory into r19-r22
                mov r21,r19
                mov r22,r20
                ldi r23,160
                ldi r24,0

                call DrawSlider                                 ;draw a slider at x-position
r21,r22 for a slider bar with top y-position in r23/r24.

        SelectBlueSliderColourEnd:
                pop r22
                pop r21
                pop r20
                pop r19
                pop r18
                pop r17
                pop r16
                ret

SelectBrushSliderSize:                                  ;returns the clicked blue value in
r19, and updates the slider value for the x coordinate to memory $0155
                push r16
                push r17
                push r18
                push r19
                push r20
                push r21
                push r22

                ldi r19,7                                       ;xlow
                ldi r20,0                                       ;xhigh
                ldi r21,$75                                     ;ylow
                ldi r22,$01                                             ;yhigh
                call MouseOnBar                         ;returns $ff if cursor on slider, else
$00

                cpi r16, $ff
                BRNE SelectBrushSliderSizeEnd   ;if r16 not equal to $ff..
                                                                                ;... NOP
                call CursorLoad                                 ;else, loads mouse position
into r19-r22
                subi r19,7                                              ;Detracts 7 from
yghigh coordinate
                lsr r19                                                 ;Divides it by 2,
since there are 32 out of 64 points available for size selection

                push ZL
                push ZH

                ldi ZL,$55                                              ;Updates slider
value in memory
```

```
                ldi ZH,$01
                ST Z, r19                                   ;Stores xlow value in Z
register

                pop ZH
                pop ZL

                call DrawBrushSizeTool                      ;Draws the brush square tool on
screen

                call CursorLoad                             ;Loads mouse position into
r19-r22
                mov r21,r19
                mov r22,r20
                ldi r23,$75
                ldi r24,$01

                call DrawSlider                             ;Draw a slider at x-position
r21,r22 for a slider bar with top y-position in r23/r24.

        SelectBrushSliderSizeEnd:                           ;NOP
                pop r22
                pop r21
                pop r20
                pop r19
                pop r18
                pop r17
                pop r16
                ret

SetSlidersToForegroundColour:                               ;Takes values for RGB sliders and sets it as
foreground colour,
                push r22
                push r23
                push r24
                push r25
                push r26
                push ZL
                push ZH

                call LoadForegroundColour                   ;Loads foreground colour
                call ThreeBytesFromColour                   ;Creates separate rgb bytes in r22-
24 from a 2 byte colour in r25/26

                ldi ZL,$57                                  ;Loads memory
pointer 0157 in register Z
                ldi ZH,$01
                ST Z+,r22                                   ;Stores r22 into
memory 0157
                ST Z+,r23                                   ;Stores r23 into
memory 0158
                ST Z+,r24                                   ;Stores r24 into
memory 0159

        ;red slider
                push r24
```

```
                push r23

                lsl r22                                         ;Multiplies by 2
000RRRRR -> 00RRRRR0
                subi r22, 249                                   ;adding 7
                mov r21, r22
                ldi r22, $00

                ldi r23, 110 ;ylow
                ldi r24, 0   ;yhigh

                call DrawSlider                                 ;draw a slider at x-position
r21,r22 for a slider bar with top y-position in r23/r24.

        ;green slider
                pop r23

                subi r23, 249                                   ;adding 7
                mov r21, r23
                ldi r22, $00

                ldi r23, 135                                    ;ylow
                ldi r24, 0                                      ;yhigh

                call DrawSlider                                 ;draw a slider at x-position
r21,r22 for a slider bar with top y-position in r23/r24.

        ;blue slider

                pop r24
                lsl r24                                         ;Multiplies by 2
000BBBBB -> 00BBBBB0
                subi r24, 249                                   ;adding 7
                mov r21, r24
                ldi r22, $00

                ldi r23, 160                                    ;ylow
                ldi r24, 0                                      ;yhigh


                call DrawSlider                                 ;draw a slider at x-position
r21,r22 for a slider bar with top y-position in r23/r24.

                pop ZH
                pop ZL
                pop r26
                pop r25
                pop r24
                pop r23
                pop r22
                ret

UpdateSavedColour:                                              ;Calculates the 2 byte
colour from the RBG slider positions, saves it to memory, and sets the foreground colour to it.
                push r17
                push r22
```

```
                push r23
                push r24
                push r25
                push r26
                push ZL
                push ZH

                ldi ZL,$56
                ldi ZH,$01
                ld r17,Z                                        ;Loads r17 with the data in
memory

                cpi r17,10                                      ;00001010
                BREQ UpdateSavedColour1
                cpi r17,11                                      ;00001011
                BREQ UpdateSavedColour2
                cpi r17,12                                      ;00001100
                BREQ UpdateSavedColour3

                jmp UpdateSavedColourEnd

        UpdateSavedColour1:
                ldi ZL,$57
                ldi ZH,$01;ColourSlider
                LD r22,Z+                                       ;$0157
                LD r23,Z+                                       ;$0158
                LD r24,Z+                                       ;$0159
                call ColourFrom3Bytes

                ldi ZL,$60
                ldi ZH,$01                                      ;Custom colours
memories, RED
                ST Z+,r25                                       ;$0160
                ST Z+,r26                                       ;$0161
                call SetForegroundColour
                call DrawColours
                call DrawBrushSizeTool
                jmp UpdateSavedColourEnd

        UpdateSavedColour2:                                     ;Loads to r22,r23,r24
ColourSlider memories and creates a colour
                ldi ZL,$57
                ldi ZH,$01
                LD r22,Z+ ;$0157, ColourSlider memories
                LD r23,Z+ ;$0158
                LD r24,Z+ ;$0159
                call ColourFrom3Bytes                   ;Creates a 2 byte colour in r25/26
from the separate RBG bytes in r22-24



                ldi ZL,$62
                ldi ZH,$01                                      ;Custom colours
memories, GREEN
                ST Z+,r25
                ST Z+,r26
```

```
                call SetForegroundColour
                call DrawColours                                ;To user interface, uploads the 12
colours squares
                call DrawBrushSizeTool                          ;Draws the brush square tool on
screen
                jmp UpdateSavedColourEnd

        UpdateSavedColour3:                                     ;Loads to r22,r23,r24
ColourSlider memories and creates a colour
                ldi ZL,$57
                ldi ZH,$01
                LD r22,Z+
                LD r23,Z+
                LD r24,Z+
                call ColourFrom3Bytes                           ;Creates a 2 byte colour in r25/26
from the separate RBG bytes in r22-24

                ldi ZL,$64
                ldi ZH,$01                                                  ;Custom colours
memories, BLUE
                ST Z+,r25
                ST Z+,r26
                call SetForegroundColour
                call DrawColours                                ;To user interface, uploads the 12
colours squares
                call DrawBrushSizeTool                          ;Draws the brush square tool on
screen

        UpdateSavedColourEnd:
                pop ZH
                pop ZL
                pop r26
                pop r25
                pop r24
                pop r23
                pop r22
                pop r17
                ret




;
; ToolMethods.asm
;
; Created: 26/11/2016 18:20:46
; Authors: AL4413, MV914
;

SelectColours:                                                  ;Selection Method -
Checks if mouse is selecting a colour square (r19,r22) and sets the foreground colour to it
                push r19
                push r20
                push r21
                push r22
                push r25
```

```
                push r26

        SelectColour1:                                  ;Checks if RED square
selected
                ldi r19, 5
                ldi r20, 0
                ldi r21, 5
                ldi r22, 0
                call MouseOnColourSquare                ;Compare mouse position to 20x20
square at position r19-r22
                cpi r16, $ff
                BRNE SelectColour2                      ;if red square not selected
-> go to green square

                ldi r19,1                               ;Colour 1 selected
                rcall SetSelectedColour                 ;Sets the SelectedColour value in
memory to the value in r19
                call DrawSelectedColour                 ;Draws a selection box around the
ColourSquareSelected square from memory

                ldi r25, COLOUR_REDL
                ldi r26, COLOUR_REDH
                call SetForegroundColour            ;Sets Foreground colour to RED

                rjmp SelectColoursEnd

        SelectColour2:                                  ;Checks if GREEN square
selected
                ldi r19, 30
                ldi r20, 0
                ldi r21, 5
                ldi r22, 0
                call MouseOnColourSquare                ;Compare mouse position to 20x20
square at position r19-r22
                cpi r16, $ff
                BRNE SelectColour3                      ;if green square not
selected -> go to blue square

                ldi r19,2                               ;Colour 2 selected
                rcall SetSelectedColour                 ;Sets the SelectedColour value in
memory to the value in r19
                call DrawSelectedColour                 ;Draws a selection box around the
ColourSquareSelected square from memory

                ldi r25, COLOUR_GREENL
                ldi r26, COLOUR_GREENH
                call SetForegroundColour

                rjmp SelectColoursEnd

        SelectColour3:                                  ;Checks if BLUE square
selected
                ldi r19, 55
                ldi r20, 0
                ldi r21, 5
                ldi r22, 0
```

```
                call MouseOnColourSquare                ;Compare mouse position to 20x20
square at position r19-r22
                cpi r16, $ff
                BRNE SelectColour4                      ;if blue square not selected
-> go to cyan square

                ldi r19,3                               ;Colour 3 selected
                rcall SetSelectedColour                 ;Sets the SelectedColour value in
memory to the value in r19
                call DrawSelectedColour                 ;Draws a selection box around the
ColourSquareSelected square from memory

                ldi r25, COLOUR_BLUEL
                ldi r26, COLOUR_BLUEH
                call SetForegroundColour

                rjmp SelectColoursEnd

        SelectColour4:                                  ;Checks if CYAN square
selected
                ldi r19, 5
                ldi r20, 0
                ldi r21, 30
                ldi r22, 0
                call MouseOnColourSquare                ;Compare mouse position to 20x20
square at position r19-r22
                cpi r16, $ff
                BRNE SelectColour5                      ;if cyan square not
selected -> go to magenta square

                ldi r19,4                               ;colour 4 selected
                rcall SetSelectedColour                 ;Sets the SelectedColour value in
memory to the value in r19
                call DrawSelectedColour                 ;Draws a selection box around the
ColourSquareSelected square from memory

                ldi r25, COLOUR_CYANL
                ldi r26, COLOUR_CYANH
                call SetForegroundColour

                rjmp SelectColoursEnd

        SelectColour5:                                  ;Checks if MAGENTA
square selected
                ldi r19, 30
                ldi r20, 0
                ldi r21, 30
                ldi r22, 0
                call MouseOnColourSquare                ;Compare mouse position to 20x20
square at position r19-r22
                cpi r16, $ff
                BRNE SelectColour6                      ;if magenta square not
selected -> go to yellow square

                ldi r19,5                               ;Colour 5 selected
```

```
                rcall SetSelectedColour                 ;Sets the SelectedColour value in
memory to the value in r19
                call DrawSelectedColour                 ;Draws a selection box around the
ColourSquareSelected square from memory

                ldi r25, COLOUR_MAGENTAL
                ldi r26, COLOUR_MAGENTAH
                call SetForegroundColour

                rjmp SelectColoursEnd

        SelectColour6:                                  ;Checks if YELLOW square
selected
                ldi r19, 55
                ldi r20, 0
                ldi r21, 30
                ldi r22, 0
                call MouseOnColourSquare                ;Compare mouse position to 20x20
square at position r19-r22
                cpi r16, $ff
                BRNE SelectColour7                      ;if yellow square not
selected -> go to white square

                ldi r19,6                               ;Colour 6 selected
                rcall SetSelectedColour                 ;Sets the SelectedColour value in
memory to the value in r19.
                call DrawSelectedColour                 ;Draws a selection box around the
ColourSquareSelected square from memory

                ldi r25, COLOUR_YELLOWL
                ldi r26, COLOUR_YELLOWH
                call SetForegroundColour

                rjmp SelectColoursEnd

        SelectColour7:                                  ;Checks if WHITE square
selected
                ldi r19, 5
                ldi r20, 0
                ldi r21, 55
                ldi r22, 0
                call MouseOnColourSquare                ;Compare mouse position to 20x20
square at position r19-r22
                cpi r16, $ff
                BRNE SelectColour8                      ;if white square not
selected -> go to grey square

                ldi r19,7                               ;Colour 7 selected
                rcall SetSelectedColour                 ;Sets the SelectedColour value in
memory to the value in r19.
                call DrawSelectedColour                 ;Draws a selection box around the
ColourSquareSelected square from memory

                ldi r25, COLOUR_WHITEL
                ldi r26, COLOUR_WHITEH
                call SetForegroundColour
```

```
                rjmp SelectColoursEnd

        SelectColour8:                                  ;Checks if GREY square
selected
                ldi r19, 30
                ldi r20, 0
                ldi r21, 55
                ldi r22, 0
                call MouseOnColourSquare                ;Compare mouse position to 20x20
square at position r19-r22
                cpi r16, $ff
                BRNE SelectColour9                      ;if grey square not selected
-> go to black square

                ldi r19,8                               ;Colour 8 selected
                rcall SetSelectedColour                 ;Sets the SelectedColour value in
memory to the value in r19.
                call DrawSelectedColour                 ;Draws a selection box around the
ColourSquareSelected square from memory

                ldi r25, COLOUR_GREYL
                ldi r26, COLOUR_GREYH
                call SetForegroundColour

                rjmp SelectColoursEnd

        SelectColour9:                                  ;Checks if BLACK square
selected
                ldi r19, 55
                ldi r20, 0
                ldi r21, 55
                ldi r22, 0
                call MouseOnColourSquare                ;Compare mouse position to 20x20
square at position r19-r22
                cpi r16, $ff
                BRNE SelectColour10                     ;if black square not
selected -> go to first custom square

                ldi r19,9                               ;Colour 9 selected
                rcall SetSelectedColour                 ;Sets the SelectedColour value in
memory to the value in r19.
                call DrawSelectedColour                 ;Draws a selection box around the
ColourSquareSelected square from memory

                ldi r25, COLOUR_BLACKL
                ldi r26, COLOUR_BLACKH
                call SetForegroundColour

                rjmp SelectColoursEnd

        SelectColour10:                                 ;Checks if FIRST CUSTOM
square selected
                ldi r19, 5
                ldi r20, 0
                ldi r21, 80
```

```
                ldi r22, 0
                call MouseOnColourSquare                    ;Compare mouse position to 20x20
square at position r19-r22
                cpi r16, $ff
                BRNE SelectColour11                         ;if first custom square not
selected -> go to second custom square

                ldi r19,10                                  ;First Custom
colour selected
                rcall SetSelectedColour                     ;Sets the SelectedColour value in
memory to the value in r19.
                call DrawSelectedColour                     ;Draws a selection box around the
ColourSquareSelected square from memory

                push ZL
                push ZH

                ;Sets the foreground colour on the driver memory $0160 and $0161 to first custom colour stored
in r25/26
                ldi ZL,$60
                ldi ZH,$01
                ld r25, Z+
                ld r26, Z+
                call SetForegroundColour

                pop ZH
                pop ZL

                call SetSlidersToForegroundColour   ;Puts slider pointers in the corresponding position for
the first custom colour

                rjmp SelectColoursEnd

        SelectColour11:                                     ;Checks if SECOND
CUSTOM square selected
                ldi r19, 30
                ldi r20, 0
                ldi r21, 80
                ldi r22, 0
                call MouseOnColourSquare
                cpi r16, $ff
                BRNE SelectColour12

                ldi r19,11                                  ;Second Custom
colour selected
                rcall SetSelectedColour                     ;Sets the SelectedColour value in
memory to the value in r19.
                call DrawSelectedColour                     ;Draws a selection box around the
ColourSquareSelected square from memory

                push ZL
                push ZH

                ;Sets the foreground colour on the driver memory $0162 and $0163 to second custom colour
stored in r25/26
                ldi ZL,$62
```

```
                ldi ZH,$01
                ld r25, Z+
                ld r26, Z+
                call SetForegroundColour

                pop ZH
                pop ZL

                call SetSlidersToForegroundColour       ;Puts slider pointers in the corresponding position for
the second custom colour

                rjmp SelectColoursEnd

        SelectColour12:                                 ;Checks if THIRD CUSTOM
square selected
                ldi r19, 55
                ldi r20, 0
                ldi r21, 80
                ldi r22, 0
                call MouseOnColourSquare
                cpi r16, $ff
                BRNE SelectColoursEnd

                ldi r19,12                              ;Third Custom
colour selected
                rcall SetSelectedColour                 ;Sets the SelectedColour value in
memory to the value in r19.
                call DrawSelectedColour                 ;Draws a selection box around the
ColourSquareSelected square from memory

                push ZL
                push ZH

                ;Sets the foreground colour on the driver memory $0164 and $0165 to third custom colour stored
in r25/26
                ldi ZL,$64
                ldi ZH,$01
                ld r25, Z+
                ld r26, Z+
                call SetForegroundColour

                pop ZH
                pop ZL

                call SetSlidersToForegroundColour       ;Puts slider pointers in the corresponding position for
the third custom colour

        SelectColoursEnd:
                pop r26
                pop r25
                pop r22
                pop r21
                pop r20
                pop r19
                ret
```

```
SetSelectedColour:                                  ;Sets the SelectedColour
value in memory to the value in r19.
                push ZL
                push ZH

                ldi ZL,$56
                ldi ZH,$01
                ST Z,r19

                pop ZH
                pop ZL
                ret



SelectTool1:                                        ;Selection method - Tool
box 1, RECTANGLE
                push r16
                push r19
                push r20
                push r21
                push r22
                push r25
                push r26
                push YL
                push YH

                ldi r19,5                           ;= x1low
                ldi r20,0                           ;= x1high
                ldi r21,185                         ;= y1low
                ldi r22,0                           ;= y1highSquare
                call MouseOnToolSquare              ;Compare mouse position to 20x20
square at position r19-r22

                cpi r16, $ff
                BRNE SelectTool1End                 ;if tool not selected, END
                                                                    ;else tool
selected
                ldi YL,$25
        ;CurrentDrawingMode memory $0125
                ldi YH,$01
                ldi r19,$01
                ST Y,r19
                rcall ResetTools                    ;Sets CurrentDrawingMode to
RectangleTool
                call DrawSelectedTool               ;Draws black square around
selected tool box

        SelectTool1End:
                pop YH
                pop YL
                pop r26
                pop r25
                pop r22
                pop r21
```

```
                pop r20
                pop r19
                pop r16
                ret

SelectTool2:                                                    ;Selection method - Tool
box 2, ELLIPSE
                push r16
                push r19
                push r20
                push r21
                push r22
                push r25
                push r26
                push YL
                push YH

                ldi r19,43                                              ;= x1low
                ldi r20,0                                       ;= x1high
                ldi r21,185                                     ;= y1low
                ldi r22,0                                       ;= y1highSquare
                call MouseOnToolSquare                  ;Compare mouse position to 20x20
square at position r19-r22

                cpi r16, $ff
                BRNE SelectTool2End                             ;if tool not selected, END
                                                                        ;else tool
selected
                ldi YL,$25
        ;CurrentDrawingMode memory $0125
                ldi YH,$01
                ldi r19,$02
                ST Y,r19
                rcall ResetTools                        ;Sets CurrentDrawingMode to
EllipseTool
                call DrawSelectedTool                   ;Draws black square around
selected tool box

        SelectTool2End:
                pop YH
                pop YL
                pop r26
                pop r25
                pop r22
                pop r21
                pop r20
                pop r19
                pop r16
                ret

SelectTool3:                                                    ;Selection method - Tool
box 3, TRIANGLE
                push r16
                push r19
                push r20
                push r21
```

```
                push r22
                push r25
                push r26
                push YL
                push YH

                ldi r19,5                                       ;= x1low
                ldi r20,0                                       ;= x1high
                ldi r21,222                                     ;= y1low
                ldi r22,0                                       ;= y1highSquare
                call MouseOnToolSquare                  ;Compare mouse position to 20x20
square at position r19-r22

                cpi r16, $ff
                BRNE SelectTool3End                             ;if tool not selected, END
                                                                        ;else tool
selected
                ldi YL,$25
        ;CurrentDrawingMode memory $0125
                ldi YH,$01
                ldi r19,$03                                             ;Sets
CurrentDrawingMode to TriangleTool
                ST Y,r19

                rcall ResetTools
                call DrawSelectedTool                   ;Draws black square around
selected tool box

        SelectTool3End:
                pop YH
                pop YL
                pop r26
                pop r25
                pop r22
                pop r21
                pop r20
                pop r19
                pop r16
                ret

SelectTool4:                                                    ;Selection method - Tool
box 4, BRUSH
                push r16
                push r19
                push r20
                push r21
                push r22
                push r25
                push r26
                push YL
                push YH

                ldi r19,43                                              ;= x1low
                ldi r20,0                                       ;= x1high
                ldi r21,222                                     ;= y1low
                ldi r22,0                                       ;= y1highSquare
```

```
                call MouseOnToolSquare                  ;Compare mouse position to 20x20
square at position r19-r22

                cpi r16, $ff
                BRNE SelectTool4End                     ;if tool not selected, END
                                                        ;else tool
selected
                ldi YL,$25
        ;CurrentDrawingMode memory $0125
                ldi YH,$01
                ldi r19,$00
                ST Y,r19                                ;Sets
CurrentDrawingMode to PaintBrush

                rcall ResetTools
                call DrawSelectedTool                   ;Draws black square around
selected tool box

        SelectTool4End:
                pop YH
                pop YL
                pop r26
                pop r25
                pop r22
                pop r21
                pop r20
                pop r19
                pop r16
                ret

SelectTool5:                                            ;Selection method - Tool
box 5, LINE
                push r16
                push r19
                push r20
                push r21
                push r22
                push r25
                push r26
                push YL
                push YH

                ldi r19,5                               ;= x1low
                ldi r20,0                               ;= x1high
                ldi r21,3                               ;= y1low
                ldi r22,1                               ;= y1highSquare
                call MouseOnToolSquare                  ;Compare mouse position to 20x20
square at position r19-r22

                cpi r16, $ff
                BRNE SelectTool5End                     ;if tool not selected, END
                                                        ;else tool
selected
                ldi YL,$25
        ;CurrentDrawingMode memory $0125
                ldi YH,$01
```

```
                ldi r19,$04
                ST Y,r19                                        ;Sets
CurrentDrawingMode to LineTool

                rcall ResetTools
                call DrawSelectedTool                   ;Draws black square around
selected tool box

        SelectTool5End:
                pop YH
                pop YL
                pop r26
                pop r25
                pop r22
                pop r21
                pop r20
                pop r19
                pop r16
                ret

SelectTool6:                                                    ;Selection method - Tool
box 6, ERASER
                push r16
                push r19
                push r20
                push r21
                push r22
                push r25
                push r26
                push YL
                push YH

                ldi r19,43                                              ;= x1low
                ldi r20,0                                       ;= x1high
                ldi r21,3                                       ;= y1low
                ldi r22,1                                       ;= y1highSquare
                call MouseOnToolSquare                  ;Compare mouse position to 20x20
square at position r19-r22

                cpi r16, $ff
                BRNE SelectTool6End                             ;if tool not selected, END
                                                                        ;else tool
selected
                ldi YL,$25
        ;CurrentDrawingMode memory $0125
                ldi YH,$01
                ldi r19,$05
                ST Y,r19                                        ;Sets
CurrentDrawingMode to EraserTool

                rcall ResetTools
                call DrawSelectedTool                   ;Draws black square around
selected tool box

        SelectTool6End:
                pop YH
```

```
                pop YL
                pop r26
                pop r25
                pop r22
                pop r21
                pop r20
                pop r19
                pop r16
                ret


ResetTools:                                                     ;Resets all Tools
to initial value
                rcall RectangleToolReset
                rcall EllipseToolReset
                rcall TriangleToolReset
                rcall LineToolReset
                ret


ResetBrush:                                                     ;Resets Brush to
initial value
                push r16
                push ZL
                push ZH

                ldi ZL,$55                                      ;Load address of
the BrushSize into Z register.
                ldi ZH,$01
                ldi r16,3
                ST Z,r16                                        ;Set the BrushSize to 3 (i.e.
circle of radius 3)

                pop ZH
                pop ZL
                pop r16
                ret

RectangleToolReset:
                push r16
                push ZL
                push ZH
                ldi ZL, $30
                ldi ZH, $01
                ldi r16, $00
                ST Z+, r16                                      ;$0130
                ST Z+, r16                                      ;$0131
                ST Z+, r16                                      ;$0132
                ST Z+, r16                                      ;$0133
                ST Z+, r16                                      ;$0134
                pop ZH
                pop ZL
                pop r16
                ret                                             ;Resets Rectangle to initial value

EllipseToolReset:
```

```
                push r16
                push ZL
                push ZH
                ldi ZL, $35
                ldi ZH, $01
                ldi r16, $00
                ST Z+, r16                                  ;$0135
                ST Z+, r16                                  ;$0136
                ST Z+, r16                                  ;$0137
                ST Z+, r16                                  ;$0138
                ST Z+, r16                                  ;$0139
                pop ZH
                pop ZL
                pop r16
                ret                                 ;Resets Ellipse to initial value

TriangleToolReset:
                push r16
                push ZL
                push ZH
                ldi ZL, $40
                ldi ZH, $01
                ldi r16, $00
                ST Z+, r16                                  ;$0141
                ST Z+, r16                                  ;$0142
                ST Z+, r16                                  ;$0143
                ST Z+, r16                                  ;$0144
                ST Z+, r16                                  ;$0145
                ST Z+, r16                                  ;$0146
                ST Z+, r16                                  ;$0147
                ST Z+, r16                                  ;$0148
                pop ZH
                pop ZL
                pop r16
                ret                                 ;Resets Triangle to initial value

LineToolReset:
                push r16
                push ZL
                push ZH
                ldi ZL, $50
                ldi ZH, $01
                ldi r16, $00
                ST Z+, r16                                  ;$0150
                ST Z+, r16                                  ;$0151
                ST Z+, r16                                  ;$0152
                ST Z+, r16                                  ;$0153
                ST Z+, r16                                  ;$0154
                pop ZH
                pop ZL
                pop r16
                ret                                 ;Resets Line to initial value

CustomColoursReset:
                push r16
                push ZL
```

```
                push ZH
                ldi ZL, $60
                ldi ZH, $01
                ldi r16, $00                                    ;set custom colours to
black.
                ST Z+, r16                                      ;$0160
                ST Z+, r16                                      ;$0161
                ST Z+, r16                                      ;$0162
                ST Z+, r16                                      ;$0163
                ST Z+, r16                                      ;$0164
                ST Z+, r16                                      ;$0165
                pop ZH
                pop ZL
                pop r16
                ret                                             ;Resets CustomColour to initial
value
```