Technical Design Document:

This is the design document for a program of drawing primitives programmed in MIPS assembly language. This document will describe the primitives and also the testing of said primitives. These primitives draw onto a bitmap display, setting the memory address of each pixel to a different colour to apply the functions wanted.

SetPixel(int x, int y, int colour):

The SetPixel primitive changes the colour of one individual pixel on the bitmap. It takes in 3 parameters: the x co-ordinate, the y co-ordinate and the colour. The x and y values are integer, the x being between 0 – 511 and the y being between 0 – 255, which is the size of the bitmap. The colour is a 32-bit integer in the form 0x00RRGGBB, where each colour is an individual value between 0 and 0xFF.

DrawLine(int x1, int y1, int x2, int y2, int colour):

The DrawLine primitive draws a line between two point on the bitmap. The function takes in 5 parameters: the x and y co-ordinates of the one end of the line, the x and y co-ordinates of the other end of the line and the colour. The x and y values are integer, the x being between 0 – 511 and the y being between 0 – 255, which is the size of the bitmap. The colour is a 32-bit integer in the form 0x00RRGGBB, where each colour is an individual value between 0 and 0xFF.

The line is drawn using “Bresenham’s line drawing algorithm” which is shown below in pseudo code. The line is drawn using the SetPixel function, with the algorithm working out which pixels need to be drawn to make the line.

```
function line(x0, y0, x1, y1)
   dx := abs(x1-x0)
   dy := abs(y1-y0)
   if x0 < x1 then sx := 1 else sx := -1
   if y0 < y1 then sy := 1 else sy := -1
   err := dx-dy

   loop
     setPixel(x0,y0)
     if x0 = x1 and y0 = y1 exit loop
     e2 := 2*err
     if e2 > -dy then
       err := err - dy
       x0 := x0 + sx
     end if
     if e2 <  dx then
       err := err + dx
       y0 := y0 + sy
     end if
   end loop
```

DrawRectangle (int x, int y, int width, int height, int colour):

The DrawRectangle primitive draws a rectangle based on a point and the width and height of the rectangle from that point. The function takes in 5 parameters: the x and y co-ordinates of the top left hand corner of the rectangle, the width and height of the rectangle and the colour. The x and y values are integer, the x being between 0 – 511 and the y being between 0 – 255, which is the size of the bitmap. The height and width are also integers, the width between 0 – 511 and the height between 0 – 255. The colour is a 32-bit integer in the form 0x00RRGGBB, where each colour is an individual value between 0 and 0xFF.

DrawPolygon (point[] point, int count, int colour):

The DrawPolygon primitive draws a polygon based on an array of points that make up the polygon, the number of sides and the colour of it. The function takes 3 parameters: an array of points, the count (number of sides) and the colour of the polygon. The array holds points which are a struct in the form:

```
struct point
{
    int x
    int y
}
```

The count (or number of sides) is an integer. The colour is a 32-bit integer in the form 0x00RRGGBB, where each colour is an individual value between 0 and 0xFF.

FillRectangle (int x, int y, int width, int height, int colour):

The FIllRectangle Primitive fills a rectangle with a solid colour based on a point and the width and height of the rectangle from that point. This function fills the rectangle with the colour specified. The filled colour does is optimised not to redraw over the outline of the rectangle, to save processing power. The function takes 5 parameters: the x and y co-ordinates of the top left hand corner of the rectangle, the width and height of the rectangle and the colour. The x and y values are integer, the x being between 0 – 511 and the y being between 0 – 255, which is the size of the bitmap. The height and width are also integers, the width between 0 – 511 and the height between 0 – 255. The colour is a 32-bit integer in the form 0x00RRGGBB, where each colour is an individual value between 0 and 0xFF.

DrawCircle (int cx, int cy, int radius, int colour):

The DrawCircle primitive draws a circle based on the centre point of that circle and the radius of the circle. The function takes in 4 parameters: The x and y co-ordinates of the centre point, the radius of the circle and the colour. The x and y co-ordinates and the radius are integers, the x being between 0 – 511 and the y and the radius being between 0 -255. The colour is a 32-bit integer in the form 0x00RRGGBB, where each colour is an individual value between 0 and 0xFF.

The circle is drawn using the “Midpoint Circle algorithm” which is shown below in pseudocode. The circle is drawn with the setpixel function, with the algorithm calculating the points which require pixels.

```
{
  int err = -radius;
  int x = radius;
  int y = 0;

  while (x >= y)
  {
    plot8points(cx, cy, x, y);

    error += y;
    ++y;
    error += y;

    if (error >= 0)
    {
      --x;
      error -= x;
      error -= x;
    }
  }
}

void plot8points(int cx, int cy, int x, int y)
{
  plot4points(cx, cy, x, y);
  if (x != y) plot4points(cx, cy, y, x);
}

void plot4points(int cx, int cy, int x, int y)
{
  setPixel(cx + x, cy + y);
  if (x != 0) setPixel(cx - x, cy + y);
  if (y != 0) setPixel(cx + x, cy - y);
  if (x != 0 && y != 0) setPixel(cx - x, cy - y); // the if was omitted as the radius is known to be non-zero
}
```

FillCircle (int cx, int cy, int radius, int colour):

The FillCircle primitive fills a circle with a solid colour based on the centre point of that circle and the radius of the circle. The function takes in 4 parameters: The x and y co-ordinates of the centre point, the radius of the circle and the colour. The x and y co-ordinates and the radius are integers, the x being between 0 – 511 and the y and the radius being between 0 -255. The colour is a 32-bit integer in the form 0x00RRGGBB, where each colour is an individual value between 0 and 0xFF.

The circle is filled using an adapted midpoint circle algorithm. The one used in my drawcircle primitive was an optimised version of the algorithm that used less registers, but that algorithm i found to have problems in adapting it to work for filling the circle. So instead I used the un-

optimised algorithm, and the changes to make it fill the circle involved replacing the setpixel calls with drawline calls, which filled the circle instead of just drawing the outline. The adapted un-optimised pseudocode algorithm is written below.

```
void rasterCircle(int x0, int y0, int radius)
{
 int f = 1 - radius;
 int ddF_x = 1;
 int ddF_y = -2 * radius;
 int x = 0;
 int y = radius;

drawline(x0 + radius, y0, x0 – radius, y0);

 while(x < y)
 {
  // ddF_x == 2 * x + 1;
  // ddF_y == -2 * y;
  // f == x*x + y*y - radius*radius + 2*x - y + 1;
  if(f >= 0)
  {
   y--;
   ddF_y += 2;
   f += ddF_y;
  }
  x++;
  ddF_x += 2;
  f += ddF_x;
  drawline(x0 + x, y0 + y, x0 – x, y0 + y);
  drawline(x0 + x, y0 – y, x0 – x, y0 – y);
  drawline(x0 + y, y0 + x, x0 – y, y0 + x);
  drawline(x0 + y, y0 – x, x0 – y, y0 – x);
 }
}
```

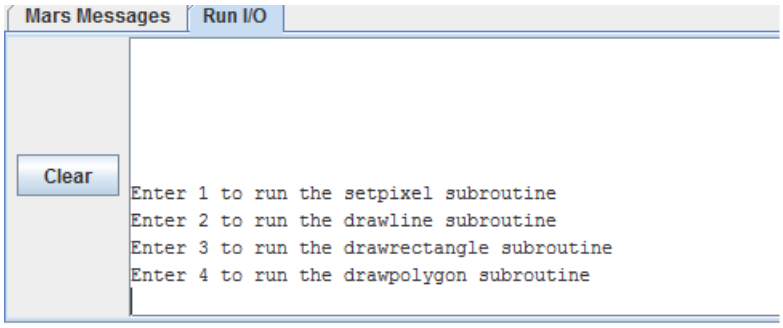
DrawDashedLine(int x1, int y1, int x2, int y2, int dash, int space, int colour):

The DrawLine primitive draws a line between two point on the bitmap. The function takes in 7 parameters: the x and y co-ordinates of the one end of the line, the x and y co-ordinates of the other end of the line, the length of the dash, the space between dhashes and the colour. The x and y values are integer, the x being between 0 – 511 and the y being between 0 – 255, which is the size of the bitmap. The length of the dash and the space between dashes are both integers. The colour is a 32-bit integer in the form 0x00RRGGBB, where each colour is an individual value between 0 and 0xFF.

The line is drawn using “Bresenham’s line drawing algorithm” which is shown above in the drawline segment.

Testing:

I have set up the program so that each primitive can be run with a variety of values to show how it works.



Test One. SetPixel Test:

This test runs the SetPixel function four times to demonstrate it. These are the four sets of values being used:

Point 1:

X co-ordinate: 200
Y co-ordinate: 100
Colour: 0x00FFFFFF (white)

Point 2:

X co-ordinate: 300
Y co-ordinate: 100
Colour: 0x00FF00FF (purple)

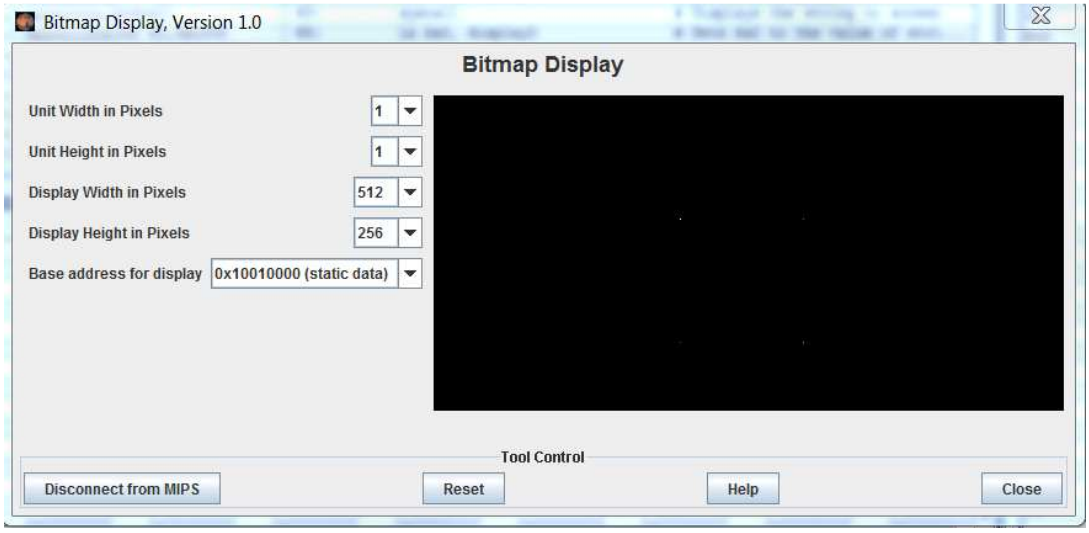
Point 3:

X co-ordinate: 300
Y co-ordinate: 200
Colour: 0x0000FF00 (green)

Point 4:

X co-ordinate: 200
Y co-ordinate: 200
Colour: 0x00FF0000 (red)

This is the bitmap with the four pixels drawn on it:



Test Two. DrawLine Test:

This test runs the DrawLine function four times to demonstrate it. These are the four sets of values being used:

Line 1:

X co-ordinate point 1: 49
Y co-ordinate point 1: 49
X co-ordinate point 2: 99
Y co-ordinate point 2: 99
Colour: 0x00FF0000 (red)

Line 2:

X co-ordinate point 1: 99
Y co-ordinate point 1: 49
X co-ordinate point 2: 199
Y co-ordinate point 2: 49
Colour: 0x00FFFFFF (white)

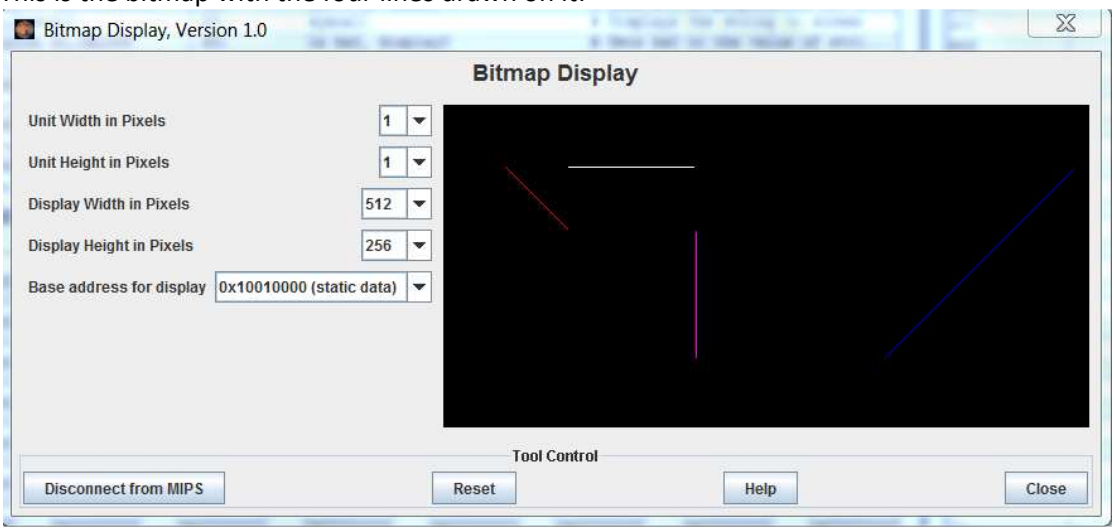
Line 3:

X co-ordinate point 1: 200
Y co-ordinate point 1: 200
X co-ordinate point 2: 200
Y co-ordinate point 2: 100
Colour: 0x00FF00FF (purple)

Line 4:

X co-ordinate point 1: 350
Y co-ordinate point 1: 200
X co-ordinate point 2: 500
Y co-ordinate point 2: 20
Colour: 0x000000FF (blue)

This is the bitmap with the four lines drawn on it:



Three. DrawRectangle Test:

This test runs the DrawRectangle function four times to demonstrate it. These are the four sets of values being used:

Rectangle 1:

X co-ordinate: 350
Y co-ordinate: 100
Width: 150
Height: 100
Colour: 0x0000FF00 (green)

Rectangle 2:

X co-ordinate: 10
Y co-ordinate: 10
Width: 100
Height: 25
Colour: 0x00FFFFFF (white)

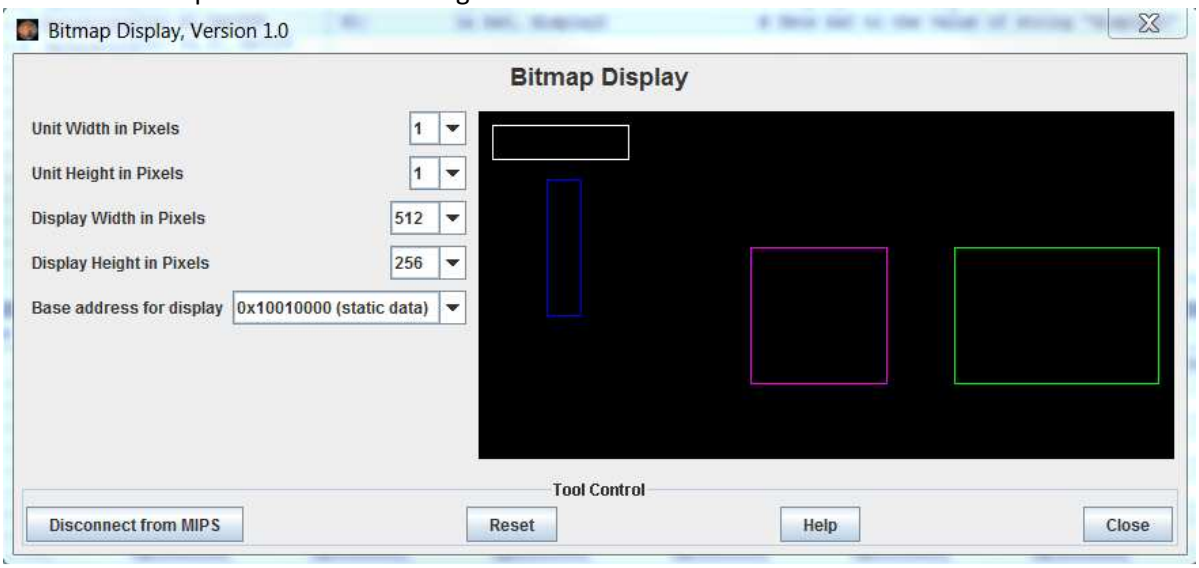
Rectangle 3:

X co-ordinate: 350
Y co-ordinate: 100
Width: 150
Height: 100
Colour: 0x0000FF00 (green)

Rectangle 4:

X co-ordinate: 350
Y co-ordinate: 100
Width: 150
Height: 100
Colour: 0x0000FF00 (green)

This is the bitmap with the four rectangles drawn on it:



Four. DrawPolygon Test:

This test runs the DrawPolygon function four times to demonstrate it. These are the four sets of values being used:

Polygon 1:

Array of points: (500, 50), (500, 150), (400, 150)
Count (number of sides): 3
Colour: 0x0000FFFF (light blue)

Polygon 2:

Array of points: (300, 200), (350, 225), (325, 250), (275, 250), (250, 225)
Count (number of sides): 5
Colour: 0x000000FF (blue)

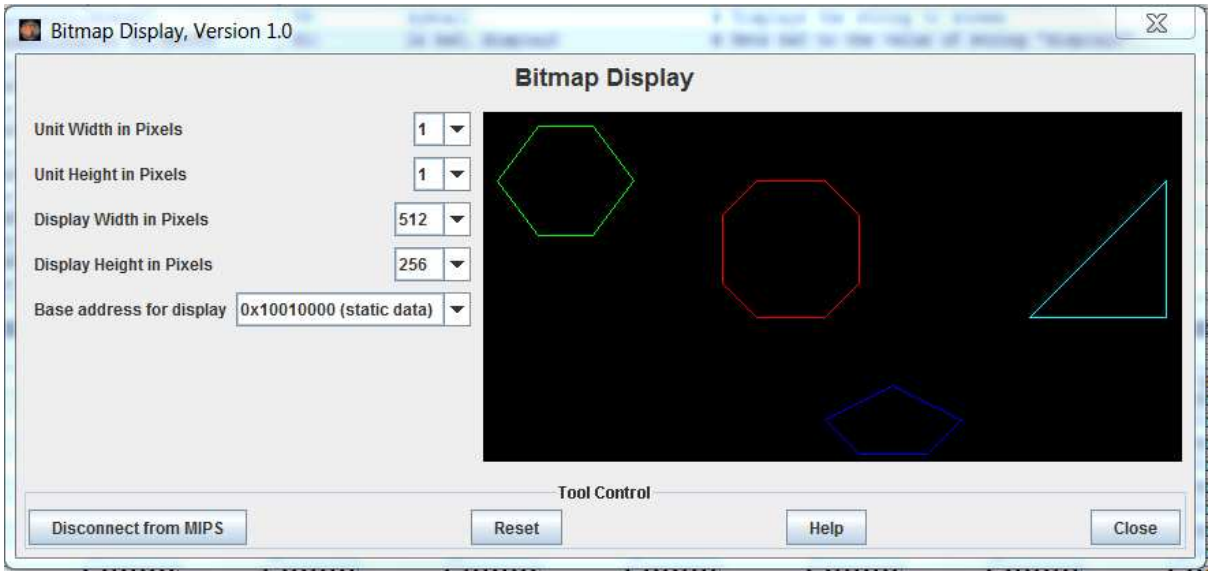
Polygon 3:

Array of points: (40, 10), (80, 10), (110, 50), (80, 90), (40, 90), (10, 50)
Count (number of sides): 6
Colour: 0x0000FF00 (green)

Polygon 4:

Array of points: (200, 50), (250, 50), (275, 75), (275, 125), (250, 150), (200, 150), (175, 125), (175, 75)
Count (number of sides): 8
Colour: 0x00FF0000 (red)

This is the bitmap with the four polygons drawn on it:



<u>Five.FillRectangle test:</u>

This test runs the FillRectangle function four times to demonstrate it. These are the four sets of values being used:

Rectangle 1:

X co-ordinate: 350
Y co-ordinate: 100
Width: 150
Height: 100
Colour: 0x0000FF00 (green)

Rectangle 2:

X co-ordinate: 10
Y co-ordinate: 10
Width: 100
Height: 25
Colour: 0x00FFFFFF (white)

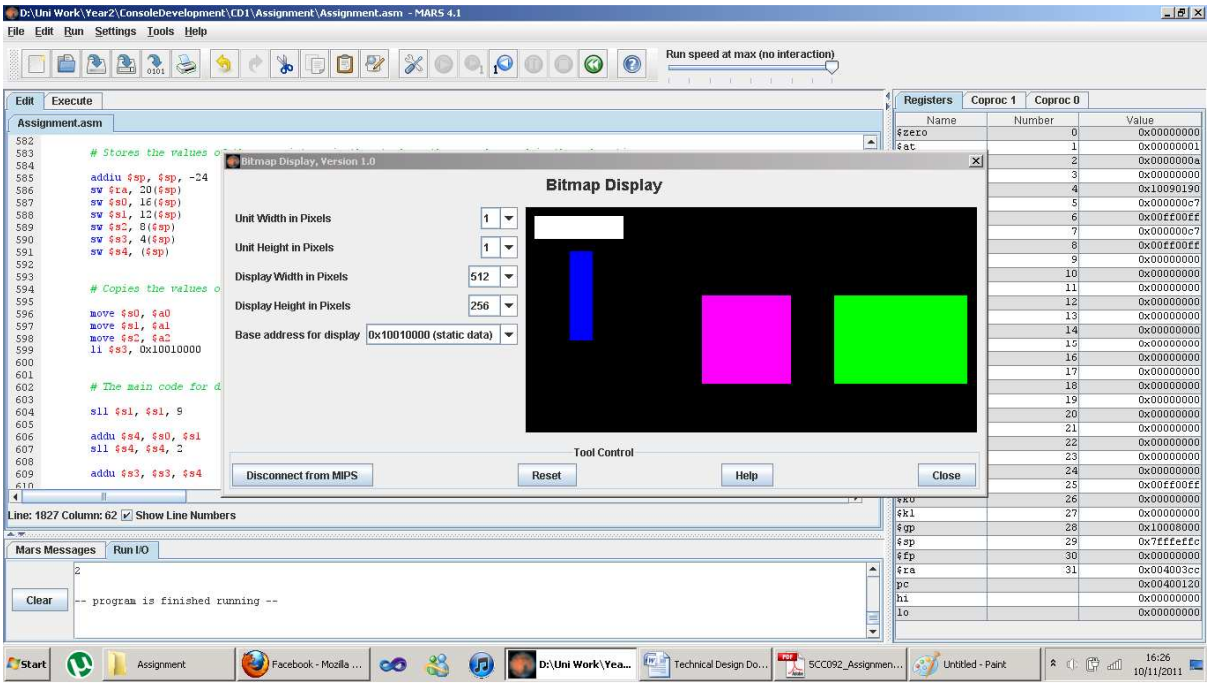
Rectangle 3:

X co-ordinate: 350
Y co-ordinate: 100
Width: 150
Height: 100
Colour: 0x0000FF00 (green)

Rectangle 4:

X co-ordinate: 350
Y co-ordinate: 100
Width: 150
Height: 100
Colour: 0x0000FF00 (green)

This is the bitmap with the four rectangles drawn on it:



Test 6. DrawCircle test:

This test runs the DrawCircle function four times to demonstrate it. These are the four sets of values being used:

Circle 1:

Centre point X co-ordinate: 50
Centre Point Y co-ordinate: 50
Radius: 25
Colour: 0x00FFFFFF (white)

Circle 2:

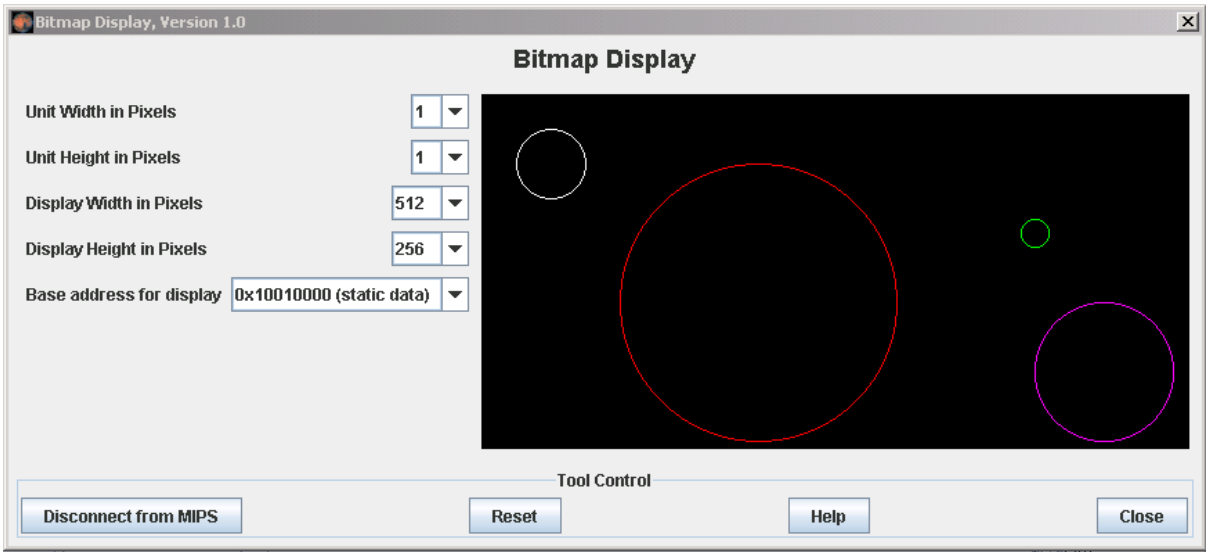
Centre point X co-ordinate: 450
Centre Point Y co-ordinate: 200
Radius: 50
Colour: 0x00FF00FF (purple)

Circle3:

Centre point X co-ordinate: 200
Centre Point Y co-ordinate: 150
Radius: 100
Colour: 0x00FF0000 (red)

Circle 4:

Centre point X co-ordinate: 400
Centre Point Y co-ordinate: 100
Radius: 10
Colour: 0x0000FF00 (green)



Test 7. FIllCircle test:

This test runs the FillCircle function four times to demonstrate it. These are the four sets of values being used:

Circle 1:

Centre point X co-ordinate: 50
Centre Point Y co-ordinate: 50
Radius: 25
Colour: 0x00FFFFFF (white)

Circle 2:

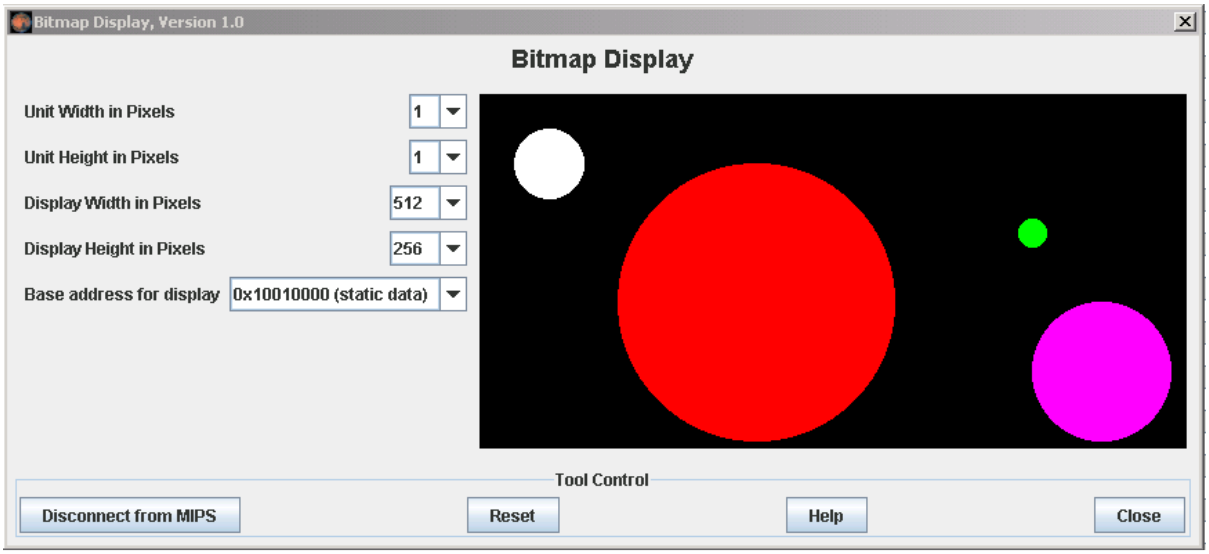
Centre point X co-ordinate: 450
Centre Point Y co-ordinate: 200
Radius: 50
Colour: 0x00FF00FF (purple)

Circle3:

Centre point X co-ordinate: 200
Centre Point Y co-ordinate: 150
Radius: 100
Colour: 0x00FF0000 (red)

Circle 4:

Centre point X co-ordinate: 400
Centre Point Y co-ordinate: 100
Radius: 10
Colour: 0x0000FF00 (green)



Two. DrawDashedLine Test:

This test runs the DrawDashedLine function four times to demonstrate it. These are the four sets of values being used:

Line 1:

X co-ordinate point 1: 49
Y co-ordinate point 1: 49
X co-ordinate point 2: 99
Y co-ordinate point 2: 99
Dash length: 1
Space length: 4
Colour: 0x00FF0000 (red)

Line 2:

X co-ordinate point 1: 99
Y co-ordinate point 1: 49
X co-ordinate point 2: 199
Y co-ordinate point 2: 49
Dash length: 1
Space length: 1
Colour: 0x00FFFFFF (white)

Line 3:

X co-ordinate point 1: 200
Y co-ordinate point 1: 200
X co-ordinate point 2: 200
Y co-ordinate point 2: 100
Dash length: 4
Space length: 1
Colour: 0x00FF00FF (purple)

Line 4:

X co-ordinate point 1: 350
Y co-ordinate point 1: 200
X co-ordinate point 2: 500
Y co-ordinate point 2: 20
Dash length: 2
Space length: 2
Colour: 0x000000FF (blue)

This is the bitmap with the four lines drawn on it: