VIDEO GAME
IN PAC
USING RFB PROTOCOL

RFB PROTOCOL

RFB or "Remote Frame Buffer" protocol was created in the Olivetti Research Laboratory to manage graphic interfaces remotely. The idea was to create a protocol as simple as possible, so as much hardware as possible (thin clients) could deal with it. One of its simplest function is to draw rectangles specifying basically 5 parameters: x position, y position, width, height and color. In that way an image can be composed of multiple rectangles, sending only 5 parameters per rectangle and there's no need to send desktop image pixel by pixel to the thin client, saving bandwidth.

RFB SERVER

The main use today of RFB servers is to take an actual image of a desktop (windows, menus, icons, apps, etc.) and send them to the remote client, also capture client actions ( mouse movements, clicks, key press) and send them to the desktop. However, due to the protocol independence from any operating system, this can be used to generate content for the client that not necessarily exists in a graphical form in the server. A classic example is a game server, where the server doesn't have screen, graphic card, controllers or keyboard, only data processing and a communication channel, it's on the client side (player) where the graphic card, monitor and keyboard are installed.
IMPLEMENTING RFB IN PAC

To develop an app RFB server two things are needed: some kind of network connectivity and some platform to write programs than can send/receive network packets. There are some examples of RFB apps, in this case a counter that can be compiled on any system with a suitable C compiler, and an extreme case where an ESP8266 module is used as an RFB server.

A PAC (in this case an Opto22 SNAP UP1 ADS) has these characteristics. If there is no PAC available, a free simulator PAC Sim can be used.

OPTOSCRIPT AND FLOWCHART PROGRAMMING

The game was developed using free PAC control for Windows suite. There isn't an official tool to program an Opto22 PAC using Linux, however, due to the control strategies sent to the PAC are text files with content coded in FORTH shouldn't be very hard to develop an alternative. The programming language uses flowcharts. Several charts could be created and run in parallel, in that way the game's logic could be split from RFB protocol implementation. A big advantage of PAC vs PLC (the gap is closes a bit every day!) is the ability to use basic network primitives, that can be used to build new protocols not included like RFB. In this case the primitives are very close to the client/server socket functions of familiar programming languages.

The game's logic, paddle positioning, ball movement, collision detection and scores were programmed in an independent flowchart, in that way the game could be accessed by other means or protocols like digital or analog I/O, or using a SCADA with Modbus, etc. The game is for a single player and there's a small "smart" algorithm that plays as a computer opponent.

More info: Absolutelyautomation.com @absolutelyautom
In the implementation of RFB protocol some shortcuts were taken, a lot of client requests are ignored, a very simple color palette was transmitted and a very crude rectangle encoding were used, so the game have not very advanced graphics!

CONCLUSIONS

• It is possible to implement the RFB protocol in an Opto22 PAC. A possible use for it is to develop "less fun" apps like remote SCADA or HMI accessible via VNC client, as a different alternative for IoT apps.

• The game only allows one single client connected, however, is possible to modify it for two or more simultaneous remote players.

• Rectangle based games where the screen doesn't scroll or changes often (i.e. Pong, Tetris, Breakout) are easier to develop using RFB.

• The control strategy works with a real PAC or with a PAC Simulator.

LINKS

Video shows two tests: Real PAC and Simulator thru VNC: http://youtu.be/rnYa2fj7JpI
Thin client for remote connection withou PC: http://s.click.aliexpress.com/e/YjA2NzZfM
GAME CHART CODE

TITLE: Chart Block Instructions
STRATEGY: RFB
CHART: game
DATE: 03/29/16  TIME: 12:35:41

______________________________

ACTIONS

Action Block: Block 0 (Id: 0)
Exit to: Init Game Vars (Id: 6)

There are no instructions in this action block.

Action Block: Game Tick Delay (Id: 1)
Exit to: Ball (Id: 9)

Delay (mSec)
GameTickDelay

Action Block: GameTick (Id: 20)
Exit to: Reset Game (Id: 25)

Move
From 1
To GameTick

SCRIPTS

OptoScript Block: Init Game Vars (Id: 6)
Enter to: Game Tick Delay (Id: 1)

GameReset = 0;
GameTickDelay = 100;
FieldColor = 0;
FieldUp = 120;
FieldDown = 480;
FieldLeft = 0;
FieldRight = 640;

BallColor = 1;
BallHeight = 11;
BallWidth = 11;

PaddleLeftColor = 2;
PaddleLeftHeight = 100;
PaddleLeftWidth = 10;
PaddleLeftposX = 30;
PaddleLeftposY = FieldUp + ( (FieldDown - FieldUp) / 2 ) ;
PaddleLeftSpeedY = 7;

PaddleRightColor = 3;
PaddleRightHeight = 100;
PaddleRightWidth = 10;
PaddleRightposX = 610;
PaddleRightposY = FieldUp + ( (FieldDown - FieldUp) / 2 ) ;
PaddleRightSpeedY = 7;

__________

More info: Absolutelyautomation.com  @absolutelyautom
BallSpeedX = 10;
BallSpeedY = 0;

BallPosX = FieldLeft + ( (FieldRight - FieldLeft) / 2 ) ;
BallPosY = FieldUp + ( (FieldDown - FieldUp) / 2 ) ;

Game7SegColor = 4;
Game7SegHeight = 30;
Game7SegWidth = 10;

GameLeft7SegAposX = 175;
GameLeft7SegAposY = 5;

GameLeft7SegBposX = 210;
GameLeft7SegBposY = 20;

GameLeft7SegCposX = 210;
GameLeft7SegCposY = 70;

GameLeft7SegDposX = 175;
GameLeft7SegDposY = 105;

GameLeft7SegEposX = 160;
GameLeft7SegEposY = 70;

GameLeft7SegFposX = 160;
GameLeft7SegFposY = 20;

GameLeft7SegGposX = 175;
GameLeft7SegGposY = 55;

GameRight7SegAposX = GameLeft7SegAposX+320;
GameRight7SegAposY = GameLeft7SegAposY;

GameRight7SegBposX = GameLeft7SegBposX+320;
GameRight7SegBposY = GameLeft7SegBposY;

GameRight7SegCposX = GameLeft7SegCposX+320;
GameRight7SegCposY = GameLeft7SegCposY;

GameRight7SegDposX = GameLeft7SegDposX+320;
GameRight7SegDposY = GameLeft7SegDposY;

GameRight7SegEposX = GameLeft7SegEposX+320;
GameRight7SegEposY = GameLeft7SegEposY;

GameRight7SegFposX = GameLeft7SegFposX+320;
GameRight7SegFposY = GameLeft7SegFposY;

GameRight7SegGposX = GameLeft7SegGposX+320;
GameRight7SegGposY = GameLeft7SegGposY;

PaddleLeftScore = 0 ;
PaddleRightScore = 0;

OptoScript Block: Ball (Id: 9)
Exit to: Paddle Left (Id: 14)

// X position
if ( BallSpeedX > 0) then
    if ( BallPosX + BallSpeedX + (BallWidth/2) > FieldRight) then
        BallPosX = FieldRight - BallWidth/2;
    else
        BallPosX = BallPosX + BallSpeedX;
    endif
endif
else
    if (BallPosX + BallSpeedX - (BallWidth/2) < FieldLeft) then
        BallPosX = FieldLeft + BallWidth/2;
    else
        BallPosX = BallPosX + BallSpeedX;
    endif
endif

// Y position

if (BallSpeedY > 0) then
    if (BallPosY + BallSpeedY + (BallHeight/2) > FieldDown) then
        BallPosY = FieldDown - BallHeight/2;
    else
        BallPosY = BallPosY + BallSpeedY;
    endif
else
    if (BallPosY + BallSpeedY - (BallHeight/2) < FieldUp) then
        BallPosY = FieldUp + BallHeight/2;
    else
        BallPosY = BallPosY + BallSpeedY;
    endif
endif

OptoScript Block: Colision Detection (Id: 10)
Exit to: GameTick (Id: 20)

// Field bounces

/*
if (BallSpeedX > 0) then
    if (BallPosX + (BallWidth/2) >= FieldRight) then
        BallSpeedX = (-1) * BallSpeedX;
        BallPosX = BallPosX -1;
    endif
endif

if (BallSpeedX < 0) then
    if (BallPosX - (BallWidth/2) <= FieldLeft) then
        BallSpeedX = (-1) * BallSpeedX;
        BallPosX = BallPosX +1;
    endif
endif
*/

if (BallSpeedX > 0) then
    if (BallPosY + (BallHeight/2) >= FieldDown) then
        BallSpeedY = (-1) * BallSpeedY;
        BallPosY = BallPosY -1;
        GameSound = 1;
    endif
endif

if (BallSpeedX < 0) then
    if (BallPosY - (BallHeight/2) <= FieldUp) then
        BallSpeedY = (-1) * BallSpeedY;
        BallPosY = BallPosY +1;
        GameSound = 1;
    endif
endif

More info: Absolutelyautomation.com @absolutelyautom
endif
endif

// Ball bounce paddle angle (BallSpeedY)
// Edge = 1.0 * BallSpeedX
// Near Edge = 0.7 * BallSpeedX
// Near center = 0.3 * BallSpeedX
// Center = 0.0 * BallSpeedX
// Near center = 0.3 * BallSpeedX
// Near Edge = 0.7 * BallSpeedX
// Edge = 1.0 * BallSpeedX

// Left paddle X zone
if ((BallPosX - (BallWidth / 2)) <= (PaddleLeftPosX + (PaddleLeftWidth / 2)) and BallSpeedX < 1) then

// edge
if (BallPosY >= (PaddleLeftPosY - (PaddleLeftHeight / 2)) AND BallPosY <= (PaddleLeftPosY - ((PaddleLeftHeight * 3) / 8))) then
if (BallSpeedY >= 0) then
    BallSpeedY = AbsoluteValue(BallSpeedX) ;
else
    BallSpeedY = AbsoluteValue(BallSpeedX) * -1 ;
endif
BallSpeedX = (-1) * BallSpeedX;
GameSound = 1;
endif
// near edge
if (BallPosY >= (PaddleLeftPosY - (PaddleLeftHeight / 4) - (PaddleLeftHeight / 8)) AND BallPosY <= (PaddleLeftPosY - (PaddleLeftHeight / 4))) then
if (BallSpeedY >= 0) then
    BallSpeedY = ( AbsoluteValue(BallSpeedX) * 2 ) / 3 ;
else
    BallSpeedY = ( AbsoluteValue(BallSpeedX) * -2 ) / 3 ;
endif
BallSpeedX = (-1) * BallSpeedX;
GameSound = 1;
endif
// Near center
if (BallPosY >= (PaddleLeftPosY - (PaddleLeftHeight / 4)) AND BallPosY <= (PaddleLeftPosY - (PaddleLeftHeight / 8))) then
if (BallSpeedY >= 0) then
    BallSpeedY = AbsoluteValue(BallSpeedX) / 3 ;
else
    BallSpeedY = AbsoluteValue(BallSpeedX) / -3 ;
endif
BallSpeedX = (-1) * BallSpeedX;
GameSound = 1;
endif
// Center
if (BallPosY >= (PaddleLeftPosY - (PaddleLeftHeight / 8)) AND BallPosY <= (PaddleLeftPosY + (PaddleLeftHeight / 8))) then
    BallSpeedY = 0;
else
    BallSpeedX = (-1) * BallSpeedX;
endif
// Near center
if (BallPosY >= (PaddleLeftPosY + (PaddleLeftHeight / 8)) AND BallPosY <= (PaddleLeftPosY + (PaddleLeftHeight / 4))) then
if (BallSpeedY >= 0) then
    BallSpeedY = AbsoluteValue(BallSpeedX) / 3 ;
else
    BallSpeedY = AbsoluteValue(BallSpeedX) / -3 ;
endif

More info: Absolutelyautomation.com @absolutelyautomic
endif

BallSpeedX = (-1) * BallSpeedX;
GameSound = 1;
endif

// near edge
if (BallPosY >= (PaddleLeftPosY + (PaddleLeftHeight/4)) AND BallPosY <= (PaddleLeftPosY + (PaddleLeftHeight/4) + (PaddleLeftHeight/8)))then
    if (BallSpeedY >= 0) then
        BallSpeedY = ( AbsoluteValue(BallSpeedX) * 2 ) /3 ;
    else
        BallSpeedY = ( AbsoluteValue(BallSpeedX) * -2 ) /3 ;
    endif

    BallSpeedX = (-1) * BallSpeedX;
    GameSound = 1;
endif

// edge
if (BallPosY >= (PaddleLeftPosY + (PaddleLeftHeight * 3 /8)) AND BallPosY <= (PaddleLeftPosY + (PaddleLeftHeight/2) ))then
    if (BallSpeedY >= 0) then
        BallSpeedY = AbsoluteValue(BallSpeedX)  ;
    else
        BallSpeedY = AbsoluteValue(BallSpeedX) * -1  ;
    endif

    BallSpeedX = (-1) * BallSpeedX;
    GameSound = 1;
endif

// Right paddle X zone
if ( (BallPosX + (BallWidth/2) ) >= (PaddleRightPosX - (PaddleRightWidth/2)) and BallSpeedX > 1) then
    // edge
    if (BallPosY >= (PaddleRightPosY - (PaddleRightHeight /2) ) AND BallPosY <= (PaddleRightPosY - ((PaddleRightHeight * 3) /8) ))then
        if (BallSpeedY >= 0) then
            BallSpeedY = AbsoluteValue(BallSpeedX) ;
        else
            BallSpeedY = AbsoluteValue(BallSpeedX) * -1 ;
        endif

        BallSpeedX = (-1) * BallSpeedX;
        GameSound = 1;
    endif

    // near edge
    if (BallPosY >= (PaddleRightPosY - (PaddleRightHeight/4)) AND BallPosY <= (PaddleRightPosY - (PaddleRightHeight/8)))then
        if (BallSpeedY >= 0) then
            BallSpeedY = ( AbsoluteValue(BallSpeedX) * 2 ) /3 ;
        else
            BallSpeedY = ( AbsoluteValue(BallSpeedX) * -2 ) /3 ;
        endif

        BallSpeedX = (-1) * BallSpeedX;
        GameSound = 1;
    endif

    // Near center
    if (BallPosY >= (PaddleRightPosY - (PaddleRightHeight/4)) AND BallPosY <= (PaddleRightPosY - (PaddleRightHeight/8))then
        if (BallSpeedY >= 0) then
            BallSpeedY = AbsoluteValue(BallSpeedX) / 3 ;
        else
            BallSpeedY = AbsoluteValue(BallSpeedX) / -3 ;
        endif
    endif

More info: Absolutelyautomation.com @absolutelyautom
BallSpeedX = (-1) * BallSpeedX;
GameSound = 1;
#endif
// Center
if( BallPosY >= (PaddleRightPosY - (PaddleRightHeight/8)) AND BallPosY <= (PaddleRightPosY + (PaddleRightHeight/8)) )then
   BallSpeedY = 0;
   BallSpeedX = (-1) * BallSpeedX;
   GameSound = 1;
endif
// Near center
if( BallPosY >= (PaddleRightPosY + (PaddleRightHeight/8)) AND BallPosY <= (PaddleRightPosY + (PaddleRightHeight/4)) )then
   if (BallSpeedY >= 0) then
      BallSpeedY = AbsoluteValue(BallSpeedX) / 3 ;
   else
      BallSpeedY = AbsoluteValue(BallSpeedX) / -3 ;
   endif
   BallSpeedX = (-1) * BallSpeedX;
   GameSound = 1;
endif
// near edge
if( BallPosY >= (PaddleRightPosY + (PaddleRightHeight/4)) AND BallPosY <= (PaddleRightPosY + (PaddleRightHeight/4) + (PaddleRightHeight/8)) )then
   if (BallSpeedY >= 0) then
      BallSpeedY = ( AbsoluteValue(BallSpeedX) * 2 ) / 3 ;
   else
      BallSpeedY = ( AbsoluteValue(BallSpeedX) * -2 ) / 3 ;
   endif
   BallSpeedX = (-1) * BallSpeedX;
   GameSound = 1;
endif
// edge
if( BallPosY >= (PaddleRightPosY + ( (PaddleRightHeight * 3 ) /8)) AND BallPosY <= (PaddleRightPosY + (PaddleRightHeight/2)) )then
   if (BallSpeedY >= 0) then
      BallSpeedY =  AbsoluteValue(BallSpeedX)  ;
   else
      BallSpeedY =  AbsoluteValue(BallSpeedX) * -1  ;
   endif
   BallSpeedX = (-1) * BallSpeedX;
   GameSound = 1;
endif
#endif
// Scores.
// Left paddle X zone
if ( (BallPosX - (BallWidth /2) ) <= PaddleLeftPosX ) then
   PaddleRightScore  = PaddleRightScore + 1;
   BallPosX = FieldLeft + ( (FieldRight - FieldLeft) / 2 ) ;
   BallPosY = FieldUp + ( (FieldDown - FieldUp) / 2 ) ;
   BallSpeedX  = AbsoluteValue(BallSpeedX) * -1 ;
   BallSpeedY = 0 ;
   GameRedrawScores = 1 ;
   GameSound = 1;
endif
// Right paddle X zone
if ( ( BallPosX + (BallWidth/2) ) >= PaddleRightPosX ) then
   PaddleLeftScore  = PaddleLeftScore + 1;
   BallPosX = FieldLeft + ( (FieldRight - FieldLeft) / 2 ) ;
   BallPosY = FieldUp + ( (FieldDown - FieldUp) / 2 ) ;
   BallSpeedX  = AbsoluteValue(BallSpeedX) * -1 ;
   BallSpeedY = 0 ;
   GameRedrawScores = 1 ;
   GameSound = 1;
endif

More info: Absolutelyautomation.com @absolutelyautom
BallPosY = FieldUp + ( (FieldDown - FieldUp) / 2 ) ;
BallSpeedX = AbsoluteValue(BallSpeedX) ;
BallSpeedY = 0 ;
GameRedrawScores = 1 ;
GameSound = 1 ;
endif

if(PaddleLeftScore > 9 or PaddleRightScore > 9)then
  PaddleLeftScore = 0 ;
  PaddleRightScore = 0 ;
endif

OptoScript Block: Paddle Left (Id: 14)
Exit to: Paddle Right (Id: 30)

// Y position
if ( PaddleLeftKey > 0) then
  if ( PaddleLeftPosY + PaddleLeftSpeedY + (PaddleLeftHeight/2) > FieldDown) then
    PaddleLeftPosY =  FieldDown - PaddleLeftHeight/2 ;
  else
    PaddleLeftPosY = PaddleLeftPosY + PaddleLeftSpeedY ;
  endif
  PaddleLeftKey = 0 ;
endif

if ( PaddleLeftKey < 0) then
  if ( PaddleLeftPosY - PaddleLeftSpeedY - (PaddleLeftHeight/2) < FieldUp) then
    PaddleLeftPosY =  FieldUp + PaddleLeftHeight/2 ;
  else
    PaddleLeftPosY = PaddleLeftPosY - PaddleLeftSpeedY ;
  endif
  PaddleLeftKey = 0 ;
endif

OptoScript Block: Paddle Right (Id: 30)
Exit to: Computer "AI" (Id: 33)

// Y position
if ( PaddleRightKey > 0) then
  if ( PaddleRightPosY + PaddleRightSpeedY + (PaddleRightHeight/2) > FieldDown) then
    PaddleRightPosY =  FieldDown - PaddleRightHeight/2 ;
  else
    PaddleRightPosY = PaddleRightPosY + PaddleRightSpeedY ;
  endif
  PaddleRightKey = 0 ;
endif

if ( PaddleRightKey < 0) then
  if ( PaddleRightPosY - PaddleRightSpeedY - (PaddleRightHeight/2) < FieldUp) then
    PaddleRightPosY =  FieldUp + PaddleRightHeight/2 ;
  else
    PaddleRightPosY = PaddleRightPosY - PaddleRightSpeedY ;
  endif
  PaddleRightKey = 0 ;
endif

More info: Absolutelyautomation.com @absolutelyautom
if (PaddleRightPosY - PaddleRightSpeedY - (PaddleRightHeight/2) < FieldUp) then
    PaddleRightPosY = FieldUp + PaddleRightHeight/2;
else
    PaddleRightPosY = PaddleRightPosY - PaddleRightSpeedY;
endif

PaddleRightKey = 0;
endif

OptoScript Block: Computer "AI" (Id: 33)
Exit to: Collision Detection (Id: 10)

// Not so smart computer intelligence!!!
if (BallPosY > PaddleRightPosY) then
    PaddleRightKey = 1;
endif
if (BallPosY < PaddleRightPosY) then
    PaddleRightKey = -1;
endif

CONDITIONS
Condition Block: Reset Game (Id: 25)
Operator Type: AND
TRUE Exit to: Init Game Vars (Id: 6)
FALSE Exit to: Game Tick Delay (Id: 1)

  Is  GameReset
  Equal?  1

CONTINUE BLOCKS
There are no continue blocks in this flowchart.
FRAMEBUFFER CHART CODE

TITL: Chart Block Instructions
STRATEGY: RFB
CHART: framebuffer
DATE: 03/29/16 TIME: 12:34:15

ACTIONS

Action Block: Block 0 (Id: 0)
Exit to: Comm handler (Id: 4)

There are no instructions in this action block.

Action Block: Listen (Id: 6)
Exit to: Accept (Id: 9)

Listen for Incoming Communication
Communication Handle COMM_TCP
Put Result in TCPstatus

Action Block: Accept (Id: 9)
Exit to: Comm Open? (Id: 11)

Accept Incoming Communication
Communication Handle COMM_TCP
Put Result in TCPstatus

Action Block: Delay 2000 (Id: 13)
Exit to: Listen (Id: 6)

Delay (mSec)
2000

Action Block: Chars waiting (Id: 39)
Exit to: Chars>0 (Id: 38)

Get Number of Characters Waiting
Communication Handle COMM_TCP
Put in nCharsWaiting

Action Block: delay (Id: 44)
Exit to: Comm Open? (Id: 18)

Delay (mSec)
10

Action Block: Chars waiting (Id: 48)
Exit to: Chars>0 (Id: 50)

Get Number of Characters Waiting
Communication Handle COMM_TCP
Put in nCharsWaiting

Action Block: delay (Id: 52)
Exit to: Comm Open? (Id: 28)

More info: Absolutelyautomation.com @absolutelyautom
Delay (mSec)
10

Action Block: Chars waiting (Id: 58)
Exit to: Chars>0 (Id: 61)

Get Number of Characters Waiting
Communication Handle COMM_TCP
Put in nCharsWaiting

Action Block: delay (Id: 63)
Exit to: Comm Open? (Id: 56)

Delay (mSec)
10

Action Block: Chars waiting (Id: 75)
Exit to: Comm Open? (Id: 138)

Get Number of Characters Waiting
Communication Handle COMM_TCP
Put in nCharsWaiting

Action Block: delay 1 ms (Id: 111)
Exit to: Game Tick (Id: 156)

Delay (mSec)
1

Action Block: Game Reset (Id: 170)
Exit to: Chars waiting (Id: 75)

Move
From 1
To GameReset

SCRIPTS

OptoScript Block: Comm handler (Id: 4)
Exit to: Listen (Id: 6)

// Init comm handler
SetCommunicationHandleValue("tcp:5900", COMM_TCP);

OptoScript Block: Send protocol version (Id: 16)
Exit to: Comm Open? (Id: 18)

TCPstringwrite = "";
TCPstringwrite = "RFB 003.003"+ chr(10);
TCPstatus = TransmitString(TCPstringwrite, COMM_TCP);

OptoScript Block: Receive client protocol version (Id: 21)
Exit to: Comm Open? (Id: 23)

More info: Absolutelyautomation.com  @absolutelyautom
TCPstatus = ReceiveNChars(TCPstringread, nCharsWaiting, COMM_TCP);

OptoScript Block: Send authentication mode (Id: 26)
Exit to: Comm Open? (Id: 28)

TCPstringwrite = "";
for tmp0 =0 to 3 step 1
  TCPstringwrite = TCPstringwrite + "X";
next

TCPstatus = SetNthCharacter(0, TCPstringwrite, 0);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 1);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 2);
TCPstatus = SetNthCharacter(1, TCPstringwrite, 3);

TCPstatus = TransmitString(TCPstringwrite, COMM_TCP);

OptoScript Block: Receive client initialisation message (Id: 31)
Exit to: Comm Open? (Id: 33)

TCPstatus = ReceiveNChars(TCPstringread, nCharsWaiting, COMM_TCP);

OptoScript Block: Send initialisation message (Id: 36)
Exit to: Comm Open? (Id: 56)

TCPstringwrite = "";
TCPstringwrite = "";
for tmp0 =0 to 27 step 1
  TCPstringwrite = TCPstringwrite + "X";
next

// Framebuffer Width
// -------------------
TCPstatus = SetNthCharacter(2, TCPstringwrite, 0);
TCPstatus = SetNthCharacter(128, TCPstringwrite, 1);

// Framebuffer Height
// -------------------
TCPstatus = SetNthCharacter(1, TCPstringwrite, 2);
TCPstatus = SetNthCharacter(224, TCPstringwrite, 3);

// Server pixel Format
// -------------------

// bits-per-pixel
TCPstatus = SetNthCharacter(8, TCPstringwrite, 4);
// dept
TCPstatus = SetNthCharacter(8, TCPstringwrite, 5);
// big endian flag
TCPstatus = SetNthCharacter(1, TCPstringwrite, 6);
// true color flag
TCPstatus = SetNthCharacter(1, TCPstringwrite, 7);
// red max
TCPstatus = SetNthCharacter(0, TCPstringwrite, 8);
TCPstatus = SetNthCharacter(7, TCPstringwrite, 9);
// green max
TCPstatus = SetNthCharacter(0, TCPstringwrite, 10);
TCPstatus = SetNthCharacter(7, TCPstringwrite, 11);
// blue max
TCPstatus = SetNthCharacter(0, TCPstringwrite, 12);
TCPstatus = SetNthCharacter(3, TCPstringwrite, 13);
// red shift
TCPstatus = SetNthCharacter(0, TCPstringwrite, 14);
// green shift

More info: Absolutelyautomation.com @absolutelyautom
TCPstatus = SetNthCharacter(3, TCPstringwrite, 15);
// blue shift
TCPstatus = SetNthCharacter(6, TCPstringwrite, 16);
// padding
TCPstatus = SetNthCharacter(0, TCPstringwrite, 17);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 18);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 19);

// name length
// --------------------
TCPstatus = SetNthCharacter(0, TCPstringwrite, 20);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 21);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 22);
TCPstatus = SetNthCharacter(4, TCPstringwrite, 23);

// name string ( name length times ...)
// --------------------
TCPstatus = SetNthCharacter(79, TCPstringwrite, 24);
TCPstatus = SetNthCharacter(80, TCPstringwrite, 25);
TCPstatus = SetNthCharacter(50, TCPstringwrite, 26);
TCPstatus = SetNthCharacter(50, TCPstringwrite, 27);

TCPstatus = TransmitString(TCPstringwrite, COMM_TCP);

OptoScript Block: Receive client message protocol (Id: 66)
Exit to: Comm Open? (Id: 69)

TCPstatus = ReceiveNChars(TCPstringread, nCharsWaiting, COMM_TCP);

OptoScript Block: Receive client requests (Id: 78)
Exit to: Searching Keyboard Messages (Id: 165)

TCPstatus = ReceiveNChars(TCPstringread, nCharsWaiting, COMM_TCP);

OptoScript Block: Send ColorMap (Id: 101)
Exit to: Initial Black Screen (Id: 108)

TCPstringwrite = "";
for tmp0 = 0 to 35 step 1
    TCPstringwrite = TCPstringwrite + "X";
next

// Color Map ( 5 colors K,R,G,B,W) easy and short
// ----------------------------------------------
TCPstatus = SetNthCharacter(1, TCPstringwrite, 0);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 1);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 2);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 3);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 4);
TCPstatus = SetNthCharacter(5, TCPstringwrite, 5);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 6);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 7);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 8);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 9);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 10);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 11);
TCPstatus = SetNthCharacter(255, TCPstringwrite, 12);

More info: Absolutelyautomation.com  @absolutelyautom
TCPstatus = SetNthCharacter(255, TCPstringwrite, 13);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 14);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 15);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 16);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 17);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 18);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 19);
TCPstatus = SetNthCharacter(255, TCPstringwrite, 20);
TCPstatus = SetNthCharacter(255, TCPstringwrite, 21);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 22);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 23);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 24);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 25);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 26);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 27);
TCPstatus = SetNthCharacter(255, TCPstringwrite, 28);
TCPstatus = SetNthCharacter(255, TCPstringwrite, 29);
TCPstatus = SetNthCharacter(255, TCPstringwrite, 30);
TCPstatus = SetNthCharacter(255, TCPstringwrite, 31);
TCPstatus = SetNthCharacter(255, TCPstringwrite, 32);
TCPstatus = SetNthCharacter(255, TCPstringwrite, 33);
TCPstatus = SetNthCharacter(255, TCPstringwrite, 34);
TCPstatus = SetNthCharacter(255, TCPstringwrite, 35);
TCPstatus = TransmitString(TCPstringwrite, COMM_TCP);

OptoScript Block: Initial Black Screen (Id: 108)
Exit to: Comm Open? (Id: 73)

TCPstringwrite = "";

for tmp0 =0 to 20 step 1
    TCPstringwrite = TCPstringwrite + "X";
next

// Framebuffer Update
// -------------------
TCPstatus = SetNthCharacter(0, TCPstringwrite, 0);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 1);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 2);
TCPstatus = SetNthCharacter(1, TCPstringwrite, 3);

// X
TCPstatus = SetNthCharacter(0, TCPstringwrite, 4);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 5);

// Y
TCPstatus = SetNthCharacter(0, TCPstringwrite, 6);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 7);

// W
TCPstatus = SetNthCharacter(2, TCPstringwrite, 8);
TCPstatus = SetNthCharacter(128, TCPstringwrite, 9);

// H
TCPstatus = SetNthCharacter(1, TCPstringwrite, 10);
TCPstatus = SetNthCharacter(224, TCPstringwrite, 11);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 12);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 13);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 14);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 15);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 16);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 17);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 18);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 19);

// pixel color data according to bits per pixel
TCPstatus = SetNthCharacter(0, TCPstringwrite, 20);

TCPstatus = TransmitString(TCPstringwrite, COMM_TCP);

OptoScript Block: Redraw Ball & Paddles (Id: 155)
Exit to: Sound (Id: 185)

GameTick = 0;
TCPstringwrite = "";
for tmp0 =0 to 105 step 1
TCPstringwrite = TCPstringwrite + "X";
next

// First time game vars non undefined
if(zFBtmp1 == 0 )then
  zFBtmp1 = BallPosX;
endif
if(zFBtmp2 == 0 )then
  zFBtmp2 = BallPosY;
endif
if(zFBtmp3 == 0 )then
  zFBtmp3 = PaddleLeftPosX;
endif
if(zFBtmp4 == 0 )then
  zFBtmp4 = PaddleLeftPosY;
endif
if(zFBtmp5 == 0 )then
  zFBtmp5 = PaddleRightPosX;
endif
if(zFBtmp6 == 0 )then
  zFBtmp6 = PaddleRightPosY;
endif

// Framebuffer Update ( ball, paddles )
// ------------------------------------
TCPstatus = SetNthCharacter(0, TCPstringwrite, 0);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 1);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 2);
TCPstatus = SetNthCharacter(6, TCPstringwrite, 3);

// Erasing previous BALL
// X

More info: Absolutelyautomation.com @absolutelyautom
int8H = ( zFBtmp1 - (BallWidth/2) ) >> 8;
int8L = 0x000000FF bitand ( zFBtmp1 - (BallWidth/2) );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 4);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 5);

// Y
int8H = ( zFBtmp2 - (BallHeight/2) ) >> 8;
int8L = 0x000000FF bitand ( zFBtmp2 - (BallHeight/2) );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 6);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 7);

// W
int8H = BallWidth >> 8;
int8L = 0x000000FF bitand BallWidth;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 8);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 9);

// H
int8H = BallHeight >> 8;
int8L = 0x000000FF bitand BallHeight;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 10);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 11);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 12);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 13);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 14);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 15);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 16);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 17);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 18);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 19);

// pixel color data according to bits per pixel
TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 20);

// Draw actual BALL

// X
int8H = ( BallPosX - (BallWidth/2) ) >> 8;
int8L = 0x000000FF bitand ( BallPosX - (BallWidth/2) );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 21);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 22);

// Y
int8H = ( BallPosY - (BallHeight/2) ) >> 8;
int8L = 0x000000FF bitand ( BallPosY - (BallHeight/2) );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 23);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 24);

// W
int8H = BallWidth >> 8;
int8L = 0x000000FF bitand BallWidth;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 25);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 26);

// H
int8H = BallHeight >> 8;
int8L = 0x000000FF bitand BallHeight;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 27);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 28);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 29);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 30);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 31);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 32);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 33);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 34);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 35);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 36);

// pixel color data according to bits per pixel
TCPstatus = SetNthCharacter(BallColor, TCPstringwrite, 37);

// Erasing previous PADDLE LEFT

// X
int8H = ( FieldLeft ) >> 8;
int8L = 0x000000FF bitand ( FieldLeft );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 38);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 39);

// Y
int8H = ( FieldUp ) >> 8;
int8L = 0x000000FF bitand ( FieldUp );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 40);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 41);

// W
int8H = ( ( PaddleLeftPosX + (PaddleLeftWidth/2) ) - FieldLeft ) >> 8;
int8L = 0x000000FF bitand ( ( PaddleLeftPosX + (PaddleLeftWidth/2) ) - FieldLeft );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 42);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 43);

// H
int8H = ( FieldDown - FieldUp ) >> 8;
int8L = 0x000000FF bitand ( FieldDown - FieldUp );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 44);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 45);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 46);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 47);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 48);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 49);

// pixel color data according to bits per pixel
TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 54);

// Draw actual PADDLE LEFT

// X
int8H = ( PaddleLeftPosX - (PaddleLeftWidth/2) ) >> 8;
int8L = 0x000000FF bitand ( PaddleLeftPosX - (PaddleLeftWidth/2) );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 55);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 56);

// Y
int8H = ( PaddleLeftPosY - (PaddleLeftHeight/2) ) >> 8;
int8L = 0x000000FF bitand ( PaddleLeftPosY - (PaddleLeftHeight/2) );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 57);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 58);

// W
int8H = PaddleLeftWidth >> 8;
int8L = 0x000000FF bitand PaddleLeftWidth;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 59);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 60);

// H
int8H = PaddleLeftHeight >> 8;
int8L = 0x000000FF bitand PaddleLeftHeight;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 61);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 62);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 63);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 64);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 65);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 66);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 67);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 68);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 69);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 70);

// pixel color data according to bits per pixel
TCPstatus = SetNthCharacter(PaddleLeftColor, TCPstringwrite, 71);

// Erasing previous PADDLE RIGHT

// X
int8H = ( PaddleRightPosX - (PaddleRightWidth/2 )) >> 8;
int8L = 0x000000FF bitand ( PaddleRightPosX - (PaddleRightWidth/2 ) );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 72);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 73);

// Y
int8H = ( FieldUp ) >> 8;
int8L = 0x000000FF bitand ( FieldUp );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 74);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 75);

// W
int8H = ( FieldRight - ( FieldRightPosX - (PaddleRightWidth/2 )) >> 8;
int8L = 0x000000FF bitand ( FieldRight - ( PaddleRightPosX - (PaddleRightWidth/2 ) ) );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 76);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 77);

// H
int8H = ( FieldDown - FieldUp ) >> 8;
int8L = 0x000000FF bitand ( FieldDown - Fieldup );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 78);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 79);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 80);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 81);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 82);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 83);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 84);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 85);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 86);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 87);

// pixel color data according to bits per pixel
TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 88);

// Draw actual PADDLE RIGHT
// X
int8H = ( PaddleRightPosX - (PaddleRightWidth/2) ) >> 8;
int8L = 0x000000FF bitand ( PaddleRightPosX - (PaddleRightWidth/2) );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 89);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 90);

// Y
int8H = ( PaddleRightPosY - (PaddleRightHeight/2) ) >> 8;
int8L = 0x000000FF bitand ( PaddleRightPosY - (PaddleRightHeight/2) );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 91);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 92);

// W
int8H = PaddleRightWidth >> 8;
int8L = 0x000000FF bitand PaddleRightWidth;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 93);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 94);

// H
int8H = PaddleRightHeight >> 8;
int8L = 0x000000FF bitand PaddleRightHeight;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 95);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 96);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 97);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 98);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 99);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 100);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 101);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 102);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 103);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 104);

// pixel color data according to bits per pixel
TCPstatus = SetNthCharacter(PaddleRightColor, TCPstringwrite, 105);

// Transmit socket RFB
TCPstatus = TransmitString(TCPstringwrite, COMM_TCP);

// Update previous positions
zFBtmp1 = BallPosX;
zFBtmp2 = BallPosY;
zFBtmp3 = PaddleLeftPosX;
zFBtmp4 = PaddleLeftPosY;
zFBtmp5 = PaddleRightPosX;
zFBtmp6 = PaddleRightPosY;

OptoScript Block: Searching Keyboard Messages (Id: 165)
Exit to: Comm Open? (Id: 85)

// Ignoring ALL client request except keyboard actions
// w key pressed 0x04 0xZZ 0xYY 0xYY 0x00 0x00 0x00 0x77
// s key pressed 0x04 0xZZ 0xYY 0xYY 0x00 0x00 0x00 0x73
// 0xZZ NON zero!
// 0xYY padding

tmp1 = 0;
tmp4 = 1;
while ( tmp4 )
    tmp0 = FindCharacterInString(4, tmp1, TCPstringread);
    if (tmp0 > -1 ) then
        tmp2 = GetNthCharacter(TCPstringread, tmp0+1);
        if(tmp2 > 0) then
            tmp3 = GetNthCharacter(TCPstringread, tmp0+7);
            if(tmp3 == 0x77) then
                PaddleLeftKey = -1;
            endif
            if(tmp3 == 0x73) then
                PaddleLeftKey = 1;
            endif
            if(tmp3 < 0) then
                tmp4 = 0;
            endif
        else
            tmp4 = 0;
        endif
        else
            tmp4 = 0;
        endif
        tmp1=tmp0+1;
wend

OptoScript Block: Redraw Scores Left (Id: 174)
Exit to: Redraw Scores Right (Id: 179)

GameRedrawScores = 0;
GameTick = 0;
TCPstringwrite = "";
for tmp0 =0 to 139 step 1
    TCPstringwrite = TCPstringwrite + "X";
next

// mask for 7 segment display to draw.
// Bit 0 seg A,
// Bit 1 seg B
//
// Bit 7 seg G

if (PaddleLeftScore == 0) then
    GameLeft7SegMask = 0xB00111111;
endif
if (PaddleLeftScore == 1) then
    GameLeft7SegMask = 0xB00000110;
endif
if (PaddleLeftScore == 2) then
    GameLeft7SegMask = 0xB01011011;
endif
if (PaddleLeftScore == 3) then
    GameLeft7SegMask = 0xB11011011;
endif
if (PaddleLeftScore == 4) then
    GameLeft7SegMask = 0xB11100111;
endif
if (PaddleLeftScore == 5) then
    GameLeft7SegMask = 0xB11011010;
endif
if (PaddleLeftScore == 6) then
    GameLeft7SegMask = 0xB11110010;
endif
if (PaddleLeftScore == 7) then
    GameLeft7SegMask = 0xB11111010;
endif
if (PaddleLeftScore == 8) then
    GameLeft7SegMask = 0xB10110111;
endif
if (PaddleLeftScore == 9) then
    GameLeft7SegMask = 0xB01110111;
endif
if (PaddleLeftScore == 10) then
    GameLeft7SegMask = 0xB01111101;
endif
if (PaddleLeftScore == 11) then
    GameLeft7SegMask = 0xB01111011;
endif
if (PaddleLeftScore == 12) then
    GameLeft7SegMask = 0xB01101011;
endif
if (PaddleLeftScore == 13) then
    GameLeft7SegMask = 0xB01011111;
endif
if (PaddleLeftScore == 14) then
    GameLeft7SegMask = 0xB01011101;
endif
if (PaddleLeftScore == 15) then
    GameLeft7SegMask = 0xB01010101;
endif
if (PaddleLeftScore == 16) then
    GameLeft7SegMask = 0xB01010001;
endif
if (PaddleLeftScore == 17) then
    GameLeft7SegMask = 0xB01001001;
endif
if (PaddleLeftScore == 18) then
    GameLeft7SegMask = 0xB01001000;
endif
if (PaddleLeftScore == 19) then
    GameLeft7SegMask = 0xB01000011;
endif
if (PaddleLeftScore == 20) then
    GameLeft7SegMask = 0xB01000000;
endif
if (PaddleLeftScore == 21) then
    GameLeft7SegMask = 0xB00100000;
endif
if (PaddleLeftScore == 22) then
    GameLeft7SegMask = 0xB00010000;
endif
More info: Absolutelyautomation.com @absolutelyautom
GameLeft7SegMask = 0xB0100111;
endif
if (PaddleLeftScore == 4) then
GameLeft7SegMask = 0xB01100110;
endif
if (PaddleLeftScore == 5) then
GameLeft7SegMask = 0xB01101101;
endif
if (PaddleLeftScore == 6) then
GameLeft7SegMask = 0xB01111101;
endif
if (PaddleLeftScore == 7) then
GameLeft7SegMask = 0xB00000111;
endif
if (PaddleLeftScore == 8) then
GameLeft7SegMask = 0xB01111111;
endif
if (PaddleLeftScore == 9) then
GameLeft7SegMask = 0xB01101111;
endif

// Framebuffer Update (scoreboard)
// ------------------------------------
TCPstatus = SetNthCharacter(0, TCPstringwrite, 0);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 1);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 2);
TCPstatus = SetNthCharacter(8, TCPstringwrite, 3);

// Erase Left scoreboard
// X
int8H = (0);
int8L = 0x000000FF bitand (0);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 4);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 5);

// Y
int8H = (0);
int8L = 0x000000FF bitand (0);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 6);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 7);

// W
int8H = 640 >> 8;
int8L = 0x000000FF bitand 320;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 8);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 9);

// H
int8H = FieldUp >> 8;
int8L = 0x000000FF bitand FieldUp;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 10);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 11);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 12);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 13);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 14);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 15);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 16);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 17);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 18);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 19);

// pixel color data according to bits per pixel
TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 20);
// Left 7 Segment A

// X
int8H = (GameLeft7SegAposX) >> 8;
int8L = 0x000000FF & GameLeft7SegAposX;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 21);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 22);

// Y
int8H = (GameLeft7SegAposY) >> 8;
int8L = 0x000000FF & GameLeft7SegAposY;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 23);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 24);

// W
int8H = Game7SegHeight >> 8;
int8L = 0x000000FF & Game7SegHeight;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 25);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 26);

// H
int8H = Game7SegWidth >> 8;
int8L = 0x000000FF & Game7SegWidth;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 27);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 28);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 29);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 30);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 31);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 32);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 33);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 34);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 35);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 36);

// pixel color data according to bits per pixel
if (GameLeft7SegMask & 0xB00000001) then
    TCPstatus = SetNthCharacter(Game7SegColor, TCPstringwrite, 37);
else
    TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 37);
endif

// Left 7 Segment B

// X
int8H = (GameLeft7SegBposX) >> 8;
int8L = 0x000000FF & GameLeft7SegBposX;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 38);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 39);

// Y
int8H = (GameLeft7SegBposY) >> 8;
int8L = 0x000000FF & GameLeft7SegBposY;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 40);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 41);

// W
int8H = Game7SegWidth >> 8;
int8L = 0x000000FF & Game7SegWidth;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 42);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 43);

// H
int8H = Game7SegHeight >> 8;
int8L = 0x000000FF & Game7SegHeight;

More info: Absolutelyautomation.com @absolutelyautum
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 44);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 45);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 46);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 47);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 48);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 49);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 50);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 51);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 52);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 53);

// pixel color data according to bits per pixel
if (GameLeft7SegMask bitand 0xB00000010) then
TCPstatus = SetNthCharacter(Game7SegColor, TCPstringwrite, 54);
else
TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 54);
endif

// Left 7 Segment C

// X
int8H = ( GameLeft7SegCposX ) >> 8;
int8L = 0x000000FF bitand ( GameLeft7SegCposX );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 55);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 56);

// Y
int8H = ( GameLeft7SegCposY ) >> 8;
int8L = 0x000000FF bitand ( GameLeft7SegCposY );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 57);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 58);

// W
int8H = Game7SegWidth >> 8;
int8L = 0x000000FF bitand Game7SegWidth;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 59);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 60);

// H
int8H = Game7SegHeight >> 8;
int8L = 0x000000FF bitand Game7SegHeight;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 61);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 62);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 63);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 64);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 65);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 66);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 67);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 68);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 69);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 70);

// pixel color data according to bits per pixel
if (GameLeft7SegMask bitand 0xB00000100) then
TCPstatus = SetNthCharacter(Game7SegColor, TCPstringwrite, 71);
else
TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 71);
endif
// Left 7 Segment D

// X
int8H = (GameLeft7SegDposX) >> 8;
int8L = 0x000000FF & (GameLeft7SegDposX);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 72);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 73);

// Y
int8H = (GameLeft7SegDposY) >> 8;
int8L = 0x000000FF & (GameLeft7SegDposY);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 74);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 75);

// W
int8H = Game7SegHeight >> 8;
int8L = 0x000000FF & Game7SegHeight;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 76);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 77);

// H
int8H = Game7SegWidth >> 8;
int8L = 0x000000FF & Game7SegWidth;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 78);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 79);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 80);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 81);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 82);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 83);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 84);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 85);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 86);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 87);

if (GameLeft7SegMask & 0xB00001000)

// pixel color data according to bits per pixel
if (GameLeft7SegColor & 0x0B000000)

More info: Absolutelyautomation.com @absolutelyautom
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 95);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 96);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 97);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 98);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 99);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 100);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 101);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 102);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 103);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 104);

// pixel color data according to bits per pixel
if (GameLeft7SegMask bitand 0xB00010000) then
  TCPstatus = SetNthCharacter(Game7SegColor, TCPstringwrite, 105);
else
  TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 105);
endif

// Left 7 Segment F

// X
int8H = ( GameLeft7SegFposX ) >> 8;
int8L = 0x000000FF bitand ( GameLeft7SegFposX );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 106);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 107);

// Y
int8H = ( GameLeft7SegFposY ) >> 8;
int8L = 0x000000FF bitand ( GameLeft7SegFposY );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 108);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 109);

// W
int8H = Game7SegWidth >> 8;
int8L = 0x000000FF bitand Game7SegWidth;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 110);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 111);

// H
int8H = Game7SegHeight >> 8;
int8L = 0x000000FF bitand Game7SegHeight;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 112);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 113);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 114);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 115);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 116);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 117);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 118);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 119);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 120);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 121);

// pixel color data according to bits per pixel
if (GameLeft7SegMask bitand 0xB00100000) then
  TCPstatus = SetNthCharacter(Game7SegColor, TCPstringwrite, 122);
else
  TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 122);
endif
// Left 7 Segment G

// X
int8H = (GameLeft7SegGposX) >> 8;
int8L = 0x000000FF & (GameLeft7SegGposX);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 123);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 124);

// Y
int8H = (GameLeft7SegGposY) >> 8;
int8L = 0x000000FF & (GameLeft7SegGposY);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 125);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 126);

// W
int8H = Game7SegHeight >> 8;
int8L = 0x000000FF & Game7SegHeight;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 127);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 128);

// H
int8H = Game7SegWidth >> 8;
int8L = 0x000000FF & Game7SegWidth;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 129);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 130);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 131);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 132);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 133);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 134);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 135);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 136);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 137);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 138);

// pixel color data according to bits per pixel
if (GameLeft7SegMask & 0xB0100000) then
    TCPstatus = SetNthCharacter(Game7SegColor, TCPstringwrite, 139);
else
    TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 139);
endif

// Transmit socket RFB
TCPstatus = TransmitString(TCPstringwrite, COMM_TCP);

OptoScript Block: Redraw Scores Right (Id: 179)
Exit to: Chars waiting (Id: 75)

GameRedrawScores = 0;
GameTick = 0;
TCPstringwrite = "";
for tmp0 = 0 to 139 step 1
    TCPstringwrite = TCPstringwrite + "X";
next
// mask for 7 segment display to draw.
// Bit 0 seg A,
if (PaddleRightScore == 0) then
    GameRight7SegMask = 0xB00111111;
endif
if (PaddleRightScore == 1) then
    GameRight7SegMask = 0xB00000110;
endif
if (PaddleRightScore == 2) then
    GameRight7SegMask = 0xB01011011;
endif
if (PaddleRightScore == 3) then
    GameRight7SegMask = 0xB01001111;
endif
if (PaddleRightScore == 4) then
    GameRight7SegMask = 0xB01100110;
endif
if (PaddleRightScore == 5) then
    GameRight7SegMask = 0xB01101101;
endif
if (PaddleRightScore == 6) then
    GameRight7SegMask = 0xB01111101;
endif
if (PaddleRightScore == 7) then
    GameRight7SegMask = 0xB00000111;
endif
if (PaddleRightScore == 8) then
    GameRight7SegMask = 0xB01111111;
endif
if (PaddleRightScore == 9) then
    GameRight7SegMask = 0xB01101111;
endif

// Framebuffer Update (scoreboard)
// ------------------------------------
TCPstatus = SetNthCharacter(0, TCPstringwrite, 0);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 1);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 2);
TCPstatus = SetNthCharacter(8, TCPstringwrite, 3);

// Erase right scoreboard

// X
int8H = 320 >> 8;
int8L = 0x000000FF bitand 320;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 4);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 5);

// Y
int8H = 0;
int8L = 0x000000FF bitand 0;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 6);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 7);

// W
int8H = 320 >> 8;
int8L = 0x000000FF bitand 320;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 8);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 9);

// H
int8H = FieldUp >> 8;
int8L = 0x000000FF bitand FieldUp;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 10);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 11);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 12);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 13);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 14);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 15);

// # sub rects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 16);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 17);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 18);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 19);

// Pixel color data according to bits per pixel
TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 20);

// Right 7 Segment A

// X
int8H = (GameRight7SegAposX) >> 8;
int8L = 0x000000FF & (GameRight7SegAposX);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 21);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 22);

// Y
int8H = (GameRight7SegAposY) >> 8;
int8L = 0x000000FF & (GameRight7SegAposY);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 23);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 24);

// W
int8H = Game7SegHeight >> 8;
int8L = 0x000000FF & Game7SegHeight;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 25);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 26);

// H
int8H = Game7SegWidth >> 8;
int8L = 0x000000FF & Game7SegWidth;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 27);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 28);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 29);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 30);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 31);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 32);

// # sub rects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 33);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 34);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 35);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 36);

// Pixel color data according to bits per pixel
if(GameRight7SegMask & 0xB00000001) then
TCPstatus = SetNthCharacter(Game7SegColor, TCPstringwrite, 37);
else
TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 37);
endif

// Right 7 Segment B

// X
int8H = (GameRight7SegBposX) >> 8;
int8L = 0x000000FF & (GameRight7SegBposX);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 38);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 39);

// Y
int8H = (GameRight7SegBposY) >> 8;
int8L = 0x000000FF bitand (GameRight7SegBposY);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 40);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 41);

// W
int8H = Game7SegWidth >> 8;
int8L = 0x000000FF bitand Game7SegWidth;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 42);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 43);

// H
int8H = Game7SegHeight >> 8;
int8L = 0x000000FF bitand Game7SegHeight;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 44);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 45);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 46);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 47);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 48);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 49);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 50);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 51);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 52);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 53);

// pixel color data according to bits per pixel
if(GameRight7SegMask bitand 0xB00000010) then
  TCPstatus = SetNthCharacter(Game7SegColor, TCPstringwrite, 54);
else
  TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 54);
endif

// Right 7 Segment C

// X
int8H = (GameRight7SegCposX) >> 8;
int8L = 0x000000FF bitand (GameRight7SegCposX);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 55);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 56);

// Y
int8H = (GameRight7SegCposY) >> 8;
int8L = 0x000000FF bitand (GameRight7SegCposY);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 57);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 58);

// W
int8H = Game7SegWidth >> 8;
int8L = 0x000000FF bitand Game7SegWidth;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 59);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 60);

// H
int8H = Game7SegHeight >> 8;
int8L = 0x000000FF bitand Game7SegHeight;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 61);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 62);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 63);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 64);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 65);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 66);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 67);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 68);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 69);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 70);

// pixel color data according to bits per pixel
if(GameRight7SegMask bitand 0xB00000100) then
  TCPstatus = SetNthCharacter(Game7SegColor, TCPstringwrite, 71);
else
  TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 71);
endif

// Right 7 Segment D

// X
int8H = ( GameRight7SegDposX ) >> 8;
int8L = 0x000000FF bitand ( GameRight7SegDposX );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 72);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 73);

// Y
int8H = ( GameRight7SegDposY ) >> 8;
int8L = 0x000000FF bitand ( GameRight7SegDposY );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 74);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 75);

// W
int8H = Game7SegHeight >> 8;
int8L = 0x000000FF bitand Game7SegHeight;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 76);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 77);

// H
int8H = Game7SegWidth >> 8;
int8L = 0x000000FF bitand Game7SegWidth;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 78);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 79);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 80);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 81);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 82);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 83);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 84);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 85);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 86);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 87);

// pixel color data according to bits per pixel
if(GameRight7SegMask bitand 0xB00001000) then
  TCPstatus = SetNthCharacter(Game7SegColor, TCPstringwrite, 88);
else
  TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 88);
endif

// Right 7 Segment E

// X
int8H = ( GameRight7SegEposX ) >> 8;
int8L = 0x000000FF bitand ( GameRight7SegEposX );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 89);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 90);

// Y

More info: Absolutelyautomation.com @absolutelyautom
int8H = ( GameRight7SegEposY ) >> 8;
int8L = 0x000000FF bitand ( GameRight7SegEposY);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 91);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 92);

// W
int8H = Game7SegWidth >> 8;
int8L = 0x000000FF bitand Game7SegWidth;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 93);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 94);

// H
int8H = Game7SegHeight >> 8;
int8L = 0x000000FF bitand Game7SegHeight;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 95);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 96);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 97);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 98);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 99);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 100);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 101);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 102);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 103);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 104);

// pixel color data according to bits per pixel
if(GameRight7SegMask bitand 0xB00010000) then
    TCPstatus = SetNthCharacter(Game7SegColor, TCPstringwrite, 105);
else
    TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 105);
endif

// Right 7 Segment F

// X
int8H = ( GameRight7SegFposX ) >> 8;
int8L = 0x000000FF bitand ( GameRight7SegFposX );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 106);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 107);

// Y
int8H = ( GameRight7SegFposY ) >> 8;
int8L = 0x000000FF bitand ( GameRight7SegFposY);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 108);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 109);

// W
int8H = Game7SegWidth >> 8;
int8L = 0x000000FF bitand Game7SegWidth;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 110);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 111);

// H
int8H = Game7SegHeight >> 8;
int8L = 0x000000FF bitand Game7SegHeight;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 112);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 113);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 114);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 115);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 116);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 117);
// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 118);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 119);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 120);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 121);

// pixel color data according to bits per pixel
if(GameRight7SegMask bitand 0xB00100000) then
    TCPstatus = SetNthCharacter(Game7SegColor, TCPstringwrite, 122);
else
    TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 122);
endif

// Right 7 Segment G

// X
int8H = ( GameRight7SegGposX ) >> 8;
int8L = 0x000000FF bitand ( GameRight7SegGposX );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 123);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 124);

// Y
int8H = ( GameRight7SegGposY ) >> 8;
int8L = 0x000000FF bitand ( GameRight7SegGposY );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 125);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 126);

// W
int8H = Game7SegHeight >> 8;
int8L = 0x000000FF bitand Game7SegHeight;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 127);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 128);

// H
int8H = Game7SegWidth >> 8;
int8L = 0x000000FF bitand Game7SegWidth;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 129);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 130);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 131);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 132);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 133);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 134);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 135);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 136);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 137);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 138);

// pixel color data according to bits per pixel
if(GameRight7SegMask bitand 0xB01000000) then
    TCPstatus = SetNthCharacter(Game7SegColor, TCPstringwrite, 139);
else
    TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 139);
endif

// Transmit socket RFB
TCPstatus = TransmitString(TCPstringwrite, COMM_TCP);

More info: Absolutelyautomation.com @absolutelyautom
// 5.1 Channel 3D sound effect
GameSound = 0;
TCPstringwrite = "";
TCPstringwrite = "X";

// Bell
// -------------------
TCPstatus = SetNthCharacter(2, TCPstringwrite, 0);
TCPstatus = TransmitString(TCPstringwrite, COMM_TCP);

// mask for 7 segment display to draw.
// Bit 0 seg A,
// Bit 1 seg B
// Bit 7 seg G
if (PaddleRightScore == 0) then
    GameRight7SegMask = 0xB00111111;
endif
if (PaddleRightScore == 1) then
    GameRight7SegMask = 0xB00000110;
endif
if (PaddleRightScore == 2) then
    GameRight7SegMask = 0xB01011011;
endif
if (PaddleRightScore == 3) then
    GameRight7SegMask = 0xB01001111;
endif
if (PaddleRightScore == 4) then
    GameRight7SegMask = 0xB01100110;
endif
if (PaddleRightScore == 5) then
    GameRight7SegMask = 0xB01101101;
endif
if (PaddleRightScore == 6) then
    GameRight7SegMask = 0xB01111101;
endif
if (PaddleRightScore == 7) then
    GameRight7SegMask = 0xB00000111;
endif
if (PaddleRightScore == 8) then
    GameRight7SegMask = 0xB01111111;
endif
if (PaddleRightScore == 9) then
    GameRight7SegMask = 0xB01101111;
endif

More info: Absolutelyautomation.com @absolutelyautom
// Framebuffer Update ( scoreboard )
// ------------------------------------
TCPstatus = SetNthCharacter(0, TCPstringwrite, 0);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 1);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 2);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 3);

// Erase right scoreboard

// X
int8H = 320 >> 8;
int8L = 0x000000FF bitand 320;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 4);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 5);

// Y
int8H = (0);
int8L = 0x000000FF bitand (0);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 6);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 7);

// W
int8H = 320 >> 8;
int8L = 0x000000FF bitand 320;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 8);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 9);

// H
int8H = FieldUp >> 8;
int8L = 0x000000FF bitand FieldUp;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 10);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 11);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 12);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 13);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 14);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 15);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 16);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 17);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 18);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 19);

// pixel color data according to bits per pixel
TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 20);

// Right 7 Segment A

// X
int8H = (GameRight7SegAposX) >> 8;
int8L = 0x000000FF bitand GameRight7SegAposX;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 21);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 22);

// Y
int8H = (GameRight7SegAposY) >> 8;
int8L = 0x000000FF bitand GameRight7SegAposY;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 23);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 24);

// W
int8H = Game7SegHeight >> 8;
int8L = 0x000000FF bitand Game7SegHeight;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 25);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 26);
```c
// H
int8H = Game7SegWidth >> 8;
int8L = 0x000000FF & Game7SegWidth;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 27);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 28);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 29);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 30);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 31);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 32);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 33);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 34);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 35);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 36);

// pixel color data according to bits per pixel
if(GameRight7SegMask & 0xB00000010) then
    TCPstatus = SetNthCharacter(Game7SegColor, TCPstringwrite, 37);
else
    TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 37);
endif

// Right 7 Segment B

// X
int8H = (GameRight7SegBposX >> 8);
int8L = 0x000000FF & GameRight7SegBposX;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 38);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 39);

// Y
int8H = (GameRight7SegBposY >> 8);
int8L = 0x000000FF & GameRight7SegBposY;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 40);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 41);

// W
int8H = Game7SegWidth >> 8;
int8L = 0x000000FF & Game7SegWidth;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 42);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 43);

// H
int8H = Game7SegHeight >> 8;
int8L = 0x000000FF & Game7SegHeight;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 44);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 45);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 46);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 47);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 48);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 49);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 50);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 51);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 52);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 53);

// pixel color data according to bits per pixel
if(GameRight7SegMask & 0xB00000010) then
    TCPstatus = SetNthCharacter(Game7SegColor, TCPstringwrite, 54);
else
```

More info: Absolutelyautomation.com   @absolutelyautom
TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 54);
endif

// Right 7 Segment C

// X
int8H = ( GameRight7SegCposX ) >> 8;
int8L = 0x000000FF bitand ( GameRight7SegCposX );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 55);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 56);

// Y
int8H = ( GameRight7SegCposY ) >> 8;
int8L = 0x000000FF bitand ( GameRight7SegCposY);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 57);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 58);

// W
int8H = Game7SegWidth >> 8;
int8L = 0x000000FF bitand Game7SegWidth;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 59);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 60);

// H
int8H = Game7SegHeight >> 8;
int8L = 0x000000FF bitand Game7SegHeight;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 61);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 62);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 63);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 64);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 65);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 66);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 67);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 68);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 69);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 70);

// pixel color data according to bits per pixel
if(GameRight7SegMask bitand 0xB00000100) then
  TCPstatus = SetNthCharacter(Game7SegColor, TCPstringwrite, 71);
else
  TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 71);
endif

// Right 7 Segment D

// X
int8H = ( GameRight7SegDposX ) >> 8;
int8L = 0x000000FF bitand ( GameRight7SegDposX );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 72);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 73);

// Y
int8H = ( GameRight7SegDposY ) >> 8;
int8L = 0x000000FF bitand ( GameRight7SegDposY);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 74);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 75);

// W
int8H = Game7SegHeight >> 8;
int8L = 0x000000FF bitand Game7SegHeight;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 76);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 77);

More info: Absolutelyautomation.com @absolutelyautom
// H
int8H = Game7SegWidth >> 8;
int8L = 0x000000FF bitand Game7SegWidth;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 78);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 79);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 80);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 81);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 82);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 83);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 84);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 85);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 86);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 87);

// pixel color data according to bits per pixel
if(GameRight7SegMask bitand 0xB00001000) then
  TCPstatus = SetNthCharacter(Game7SegColor, TCPstringwrite, 88);
else
  TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 88);
endif

// Right 7 Segment E

// X
int8H = (GameRight7SegEposX) >> 8;
int8L = 0x000000FF bitand (GameRight7SegEposX);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 89);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 90);

// Y
int8H = (GameRight7SegEposY) >> 8;
int8L = 0x000000FF bitand (GameRight7SegEposY);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 91);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 92);

// W
int8H = Game7SegWidth >> 8;
int8L = 0x000000FF bitand Game7SegWidth;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 93);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 94);

// H
int8H = Game7SegHeight >> 8;
int8L = 0x000000FF bitand Game7SegHeight;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 95);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 96);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 97);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 98);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 99);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 100);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 101);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 102);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 103);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 104);

// pixel color data according to bits per pixel
if(GameRight7SegMask bitand 0xB00010000) then
  TCPstatus = SetNthCharacter(Game7SegColor, TCPstringwrite, 105);
else
  TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 105);
endif
else
    TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 105);
endif

// Right 7 Segment F

// X
int8H = (GameRight7SegFposX) >> 8;
int8L = 0x000000FF bitand (GameRight7SegFposX);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 106);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 107);

// Y
int8H = (GameRight7SegFposY) >> 8;
int8L = 0x000000FF bitand (GameRight7SegFposY);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 108);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 109);

// W
int8H = Game7SegWidth >> 8;
int8L = 0x000000FF bitand Game7SegWidth;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 110);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 111);

// H
int8H = Game7SegHeight >> 8;
int8L = 0x000000FF bitand Game7SegHeight;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 112);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 113);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 114);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 115);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 116);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 117);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 118);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 119);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 120);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 121);

// pixel color data according to bits per pixel
if(GameRight7SegMask bitand 0xB00100000) then
    TCPstatus = SetNthCharacter(Game7SegColor, TCPstringwrite, 122);
else
    TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 122);
endif

// Right 7 Segment G

// X
int8H = (GameRight7SegGposX) >> 8;
int8L = 0x000000FF bitand (GameRight7SegGposX);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 123);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 124);

// Y
int8H = (GameRight7SegGposY) >> 8;
int8L = 0x000000FF bitand (GameRight7SegGposY);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 125);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 126);

// W
int8H = Game7SegHeight >> 8;
int8L = 0x000000FF bitand Game7SegHeight;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 127);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 128);

// H
int8H = Game7SegWidth >> 8;
int8L = 0x000000FF bitand Game7SegWidth;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 129);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 130);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 131);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 132);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 133);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 134);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 135);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 136);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 137);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 138);

// pixel color data according to bits per pixel
if (GameRight7SegMask bitand 0xB01000000) then
    TCPstatus = SetNthCharacter(Game7SegColor, TCPstringwrite, 139);
else
    TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 139);
endif

// Transmit socket RFB
TCPstatus = TransmitString(TCPstringwrite, COMM_TCP);

OptoScript Block: Redraw Scores Left (Id: 197)
Exit to: Redraw Scores Right (Id: 196)

GameRedrawScores = 0;
GameTick = 0;
TCPstringwrite = "";
for tmp0 = 0 to 139 step 1
    TCPstringwrite = TCPstringwrite + "X";
next

// mask for 7 segment display to draw.
// Bit 0 seg A,
// Bit 1 seg B
//
// Bit 7 seg G
if (PaddleLeftScore == 0) then
    GameLeft7SegMask = 0xB00011111;
endif
if (PaddleLeftScore == 1) then
    GameLeft7SegMask = 0xB00000110;
endif
if (PaddleLeftScore == 2) then
    GameLeft7SegMask = 0xB01011011;
endif
if (PaddleLeftScore == 3) then
    GameLeft7SegMask = 0xB01001111;
endif

More info: Absolutelyautomation.com @absolutelyautom
if (PaddleLeftScore == 4) then
    GameLeft7SegMask = 0xB01100110;
endif
if (PaddleLeftScore == 5) then
    GameLeft7SegMask = 0xB01101101;
endif
if (PaddleLeftScore == 6) then
    GameLeft7SegMask = 0xB01111101;
endif
if (PaddleLeftScore == 7) then
    GameLeft7SegMask = 0xB00000111;
endif
if (PaddleLeftScore == 8) then
    GameLeft7SegMask = 0xB01111111;
endif
if (PaddleLeftScore == 9) then
    GameLeft7SegMask = 0xB01101111;
endif

// Framebuffer Update (scoreboard)
// ------------------------------------
TCPstatus = SetNthCharacter(0, TCPstringwrite, 0);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 1);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 2);
TCPstatus = SetNthCharacter(8, TCPstringwrite, 3);

// Erase Left scoreboard

// X
int8H = (0);
int8L = 0x000000FF bitand (0);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 4);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 5);

// Y
int8H = (0);
int8L = 0x000000FF bitand (0);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 6);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 7);

// W
int8H = 640 >> 8;
int8L = 0x000000FF bitand 320;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 8);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 9);

// H
int8H = FieldUp >> 8;
int8L = 0x000000FF bitand FieldUp;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 10);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 11);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 12);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 13);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 14);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 15);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 16);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 17);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 18);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 19);

// pixel color data according to bits per pixel
TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 20);

// Left 7 Segment A
// X
int8H = ( GameLeft7SegAposX ) >> 8;
int8L = 0x000000FF bitand ( GameLeft7SegAposX );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 21);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 22);

// Y
int8H = ( GameLeft7SegAposY ) >> 8;
int8L = 0x000000FF bitand ( GameLeft7SegAposY );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 23);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 24);

// W
int8H = Game7SegHeight >> 8;
int8L = 0x000000FF bitand Game7SegHeight;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 25);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 26);

// H
int8H = Game7SegWidth >> 8;
int8L = 0x000000FF bitand Game7SegWidth;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 27);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 28);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 29);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 30);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 31);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 32);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 33);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 34);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 35);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 36);

// pixel color data according to bits per pixel
if (GameLeft7SegMask bitand 0xB00000001) then
    TCPstatus = SetNthCharacter(Game7SegColor, TCPstringwrite, 37);
else
    TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 37);
endif

// Left 7 Segment B

// X
int8H = ( GameLeft7SegBposX ) >> 8;
int8L = 0x000000FF bitand ( GameLeft7SegBposX );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 38);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 39);

// Y
int8H = ( GameLeft7SegBposY ) >> 8;
int8L = 0x000000FF bitand ( GameLeft7SegBposY );
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 40);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 41);

// W
int8H = Game7SegWidth >> 8;
int8L = 0x000000FF bitand Game7SegWidth;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 42);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 43);

// H
int8H = Game7SegHeight >> 8;
int8L = 0x000000FF bitand Game7SegHeight;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 44);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 45);
// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 46);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 47);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 48);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 49);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 50);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 51);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 52);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 53);

// pixel color data according to bits per pixel
if (GameLeft7SegMask bitand 0xB00000100) then
  TCPstatus = SetNthCharacter(Game7SegColor, TCPstringwrite, 54);
else
  TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 54);
endif

// Left 7 Segment C

// X
int8H = (GameLeft7SegCposX ) >> 8;
int8L = 0x000000FF bitand (GameLeft7SegCposX);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 55);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 56);

// Y
int8H = (GameLeft7SegCposY ) >> 8;
int8L = 0x000000FF bitand (GameLeft7SegCposY);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 57);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 58);

// W
int8H = Game7SegWidth >> 8;
int8L = 0x000000FF bitand Game7SegWidth;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 59);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 60);

// H
int8H = Game7SegHeight >> 8;
int8L = 0x000000FF bitand Game7SegHeight;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 61);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 62);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 63);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 64);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 65);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 66);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 67);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 68);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 69);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 70);

// pixel color data according to bits per pixel
if (GameLeft7SegMask bitand 0xB000000100) then
  TCPstatus = SetNthCharacter(Game7SegColor, TCPstringwrite, 71);
else
  TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 71);
endif

// Left 7 Segment D

More info: Absolutelyautomation.com @absolutelyautom
// X
int8H = (GameLeft7SegDposX) >> 8;
int8L = 0x000000FF bitand (GameLeft7SegDposX);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 72);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 73);

// Y
int8H = (GameLeft7SegDposY) >> 8;
int8L = 0x000000FF bitand (GameLeft7SegDposY);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 74);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 75);

// W
int8H = Game7SegHeight >> 8;
int8L = 0x000000FF bitand Game7SegHeight;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 76);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 77);

// H
int8H = Game7SegWidth >> 8;
int8L = 0x000000FF bitand Game7SegWidth;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 78);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 79);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 80);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 81);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 82);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 83);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 84);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 85);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 86);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 87);

// pixel color data according to bits per pixel
if (GameLeft7SegMask bitand 0xB00001000) then
  TCPstatus = SetNthCharacter(Game7SegColor, TCPstringwrite, 88);
else
  TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 88);
endif

// Left 7 Segment E

// X
int8H = (GameLeft7SegEposX) >> 8;
int8L = 0x000000FF bitand (GameLeft7SegEposX);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 89);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 90);

// Y
int8H = (GameLeft7SegEposY) >> 8;
int8L = 0x000000FF bitand (GameLeft7SegEposY);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 91);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 92);

// W
int8H = Game7SegWidth >> 8;
int8L = 0x000000FF bitand Game7SegWidth;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 93);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 94);

// H
int8H = Game7SegHeight >> 8;
int8L = 0x000000FF bitand Game7SegHeight;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 95);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 96);
// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 97);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 98);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 99);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 100);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 101);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 102);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 103);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 104);

// pixel color data according to bits per pixel
if (GameLeft7SegMask bitand 0xB00010000) then
    TCPstatus = SetNthCharacter(Game7SegColor, TCPstringwrite, 105);
else
    TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 105);
endif

// Left 7 Segment F

// X
int8H = (GameLeft7SegFposX) >> 8;
int8L = 0x000000FF bitand (GameLeft7SegFposX);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 106);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 107);

// Y
int8H = (GameLeft7SegFposY) >> 8;
int8L = 0x000000FF bitand (GameLeft7SegFposY);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 108);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 109);

// W
int8H = Game7SegWidth >> 8;
int8L = 0x000000FF bitand Game7SegWidth;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 110);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 111);

// H
int8H = Game7SegHeight >> 8;
int8L = 0x000000FF bitand Game7SegHeight;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 112);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 113);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 114);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 115);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 116);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 117);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 118);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 119);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 120);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 121);

// pixel color data according to bits per pixel
if (GameLeft7SegMask bitand 0xB00100000) then
    TCPstatus = SetNthCharacter(Game7SegColor, TCPstringwrite, 122);
else
    TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 122);
endif

// Left 7 Segment G

More info: Absolutelyautomation.com @absolutelyautom
// X
int8H = (GameLeft7SegGposX) >> 8;
int8L = 0x000000FF & (GameLeft7SegGposX);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 123);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 124);

// Y
int8H = (GameLeft7SegGposY) >> 8;
int8L = 0x000000FF & (GameLeft7SegGposY);
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 125);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 126);

// W
int8H = Game7SegHeight >> 8;
int8L = 0x000000FF & Game7SegHeight;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 127);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 128);

// H
int8H = Game7SegWidth >> 8;
int8L = 0x000000FF & Game7SegWidth;
TCPstatus = SetNthCharacter(int8H, TCPstringwrite, 129);
TCPstatus = SetNthCharacter(int8L, TCPstringwrite, 130);

// Enc
TCPstatus = SetNthCharacter(0, TCPstringwrite, 131);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 132);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 133);
TCPstatus = SetNthCharacter(2, TCPstringwrite, 134);

// # subrects
TCPstatus = SetNthCharacter(0, TCPstringwrite, 135);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 136);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 137);
TCPstatus = SetNthCharacter(0, TCPstringwrite, 138);

// pixel color data according to bits per pixel
if (GameLeft7SegMask & 0xB01000000) then
    TCPstatus = SetNthCharacter(Game7SegColor, TCPstringwrite, 139);
else
    TCPstatus = SetNthCharacter(FieldColor, TCPstringwrite, 139);
endif

// Transmit socket RFB
TCPstatus = TransmitString(TCPstringwrite, COMM_TCP);

CONDITIONS

Condition Block: Comm Open? (Id: 11)
Operator Type: AND
TRUE Exit to: Send protocol version (Id: 16)
FALSE Exit to: Delay 2000 (Id: 13)

Communication HandleCOMM_TCP
Communication Open?

Condition Block: Comm Open? (Id: 18)
Operator Type: AND
TRUE Exit to: Chars waiting (Id: 39)
FALSE Exit to: Delay 2000 (Id: 13)
Communication Handle COMM_TCP
Communication Open?

Condition Block: Comm Open? (Id: 23)
Operator Type: AND
TRUE Exit to: Send authentication mode (Id: 26)
FALSE Exit to: Delay 2000 (Id: 13)

Communication Handle COMM_TCP
Communication Open?

Condition Block: Comm Open? (Id: 28)
Operator Type: AND
TRUE Exit to: Chars waiting (Id: 48)
FALSE Exit to: Delay 2000 (Id: 13)

Communication Handle COMM_TCP
Communication Open?

Condition Block: Comm Open? (Id: 33)
Operator Type: AND
TRUE Exit to: Send initialisation message (Id: 36)
FALSE Exit to: Delay 2000 (Id: 13)

Communication Handle COMM_TCP
Communication Open?

Condition Block: Chars>0 (Id: 38)
Operator Type: AND
TRUE Exit to: Receive client protocol version (Id: 21)
FALSE Exit to: delay (Id: 44)

Is nCharsWaiting
Greater? Than 0

Condition Block: Chars>0 (Id: 50)
Operator Type: AND
TRUE Exit to: Receive client initialisation message (Id: 31)
FALSE Exit to: delay (Id: 52)

Is nCharsWaiting
Greater? Than 0

Condition Block: Comm Open? (Id: 56)
Operator Type: AND
TRUE Exit to: Chars waiting (Id: 58)
FALSE Exit to: Delay 2000 (Id: 13)

Communication Handle COMM_TCP
Communication Open?

Condition Block: Chars>0 (Id: 61)
Operator Type: AND
TRUE Exit to: Receive client message protocol (Id: 66)
FALSE Exit to: delay (Id: 63)
Is nCharsWaiting Greater? Than 0

Condition Block: Comm Open? (Id: 69)
Operator Type: AND
TRUE Exit to: Send ColorMap (Id: 101)
FALSE Exit to: Delay 2000 (Id: 13)

Communication HandleCOMM_TCP
Communication Open?

Condition Block: Comm Open? (Id: 73)
Operator Type: AND
TRUE Exit to: Redraw Scores Left (Id: 197)
FALSE Exit to: Delay 2000 (Id: 13)

Communication HandleCOMM_TCP
Communication Open?

Condition Block: Chars>0 (Id: 79)
Operator Type: AND
TRUE Exit to: Receive client requests (Id: 78)
FALSE Exit to: delay 1 ms (Id: 111)

Is nCharsWaiting Greater? Than 0

Condition Block: Comm Open? (Id: 85)
Operator Type: AND
TRUE Exit to: delay 1 ms (Id: 111)
FALSE Exit to: Delay 2000 (Id: 13)

Communication HandleCOMM_TCP
Communication Open?

Condition Block: Comm Open? (Id: 138)
Operator Type: AND
TRUE Exit to: Chars>0 (Id: 79)
FALSE Exit to: Delay 2000 (Id: 13)

Communication HandleCOMM_TCP
Communication Open?

Condition Block: Game Tick (Id: 156)
Operator Type: AND
TRUE Exit to: Redraw Ball & Paddles (Id: 155)
FALSE Exit to: Chars waiting (Id: 75)

Is GameTick Equal? To 1

Condition Block: Scores (Id: 173)
Operator Type: AND
TRUE Exit to: Redraw Scores Left (Id: 174)
FALSE Exit to: Chars waiting (Id: 75)

Is GameRedrawScores Equal? To 1

Condition Block: Sound (Id: 185)
Operator Type: AND
TRUE Exit to: Send Bell (Id: 182)
FALSE Exit to: Scores (Id: 173)

Is GameSound Equal? To 1

CONTINUE BLOCKS
There are no continue blocks in this flowchart.

More info: Absolutelyautomation.com @absolutelyautom
### VARIABLES

**TITLE:** Strategy Database  
**STRATEGY:** RFB  
**DATE:** 03/29/16  **TIME:** 12:33:29

---

#### NUMERIC VARIABLES

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>INIT. VALUE</th>
<th>REF. COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>BallColor</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>BallHeight</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>BallPosX</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>BallPosY</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>BallSpeedX</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>BallSpeedY</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>BallWidth</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>chartstatus</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>FieldColor</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>FieldDown</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>FieldLeft</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>FieldRight</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>Game7SegColor</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>Game7SegHeight</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>Game7SegWidth</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameLeft7SegAposX</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameLeft7SegAposY</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameLeft7SegBposX</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameLeft7SegBposY</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameLeft7SegCposX</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameLeft7SegCposY</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameLeft7SegDposX</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameLeft7SegDposY</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameLeft7SegEposX</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameLeft7SegEposY</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameLeft7SegFposX</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameLeft7SegFposY</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameLeft7SegGposX</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameLeft7SegGposY</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameLeft7SegMask</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameRedrawScores</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameReset</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameRight7SegAposX</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameRight7SegAposY</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameRight7SegBposX</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameRight7SegBposY</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameRight7SegCposX</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameRight7SegCposY</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameRight7SegDposX</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameRight7SegDposY</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameRight7SegEposX</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameRight7SegEposY</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameRight7SegFposX</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameRight7SegFposY</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameRight7SegGposX</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameRight7SegGposY</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameRight7SegMask</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameSound</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameTick</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>GameTickDelay</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>int8H</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>init8L</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>nCharsWaiting</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>PaddleLeftColor</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>PaddleLeftHeight</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
<tr>
<td>PaddleLeftKey</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
</tr>
</tbody>
</table>

**More info:** Absolutelyautomation.com  
@absolutelyautom
<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Type</th>
<th>Access</th>
<th>Width</th>
<th>Ref.</th>
<th>Count</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PaddleLeftPosX</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PaddleLeftPosY</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PaddleLeftScore</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PaddleLeftSpeedY</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PaddleLeftWidth</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PaddleRightColor</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PaddleRightHeight</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PaddleRightKey</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PaddleRightPosX</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PaddleRightPosY</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
<td>29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PaddleRightScore</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PaddleRightSpeedY</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PaddleRightWidth</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCPstatus</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
<td>801</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tmp0</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tmp1</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tmp2</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tmp3</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tmp4</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>zFBtmp1</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>zFBtmp2</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>zFBtmp3</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>zFBtmp4</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>zFBtmp5</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>zFBtmp6</td>
<td>INT</td>
<td>RUN</td>
<td>0</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**STRING VARIABLES**

<table>
<thead>
<tr>
<th>Name</th>
<th>Init. Width</th>
<th>Ref. Count</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCPstringread</td>
<td>RUN 512</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>TCPstringwrite</td>
<td>RUN 512</td>
<td>827</td>
<td></td>
</tr>
</tbody>
</table>

**COMMUNICATION HANDLES**

<table>
<thead>
<tr>
<th>Name</th>
<th>Ref. Count</th>
<th>Init. Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMM_TCP</td>
<td>32</td>
<td>RUN</td>
</tr>
</tbody>
</table>