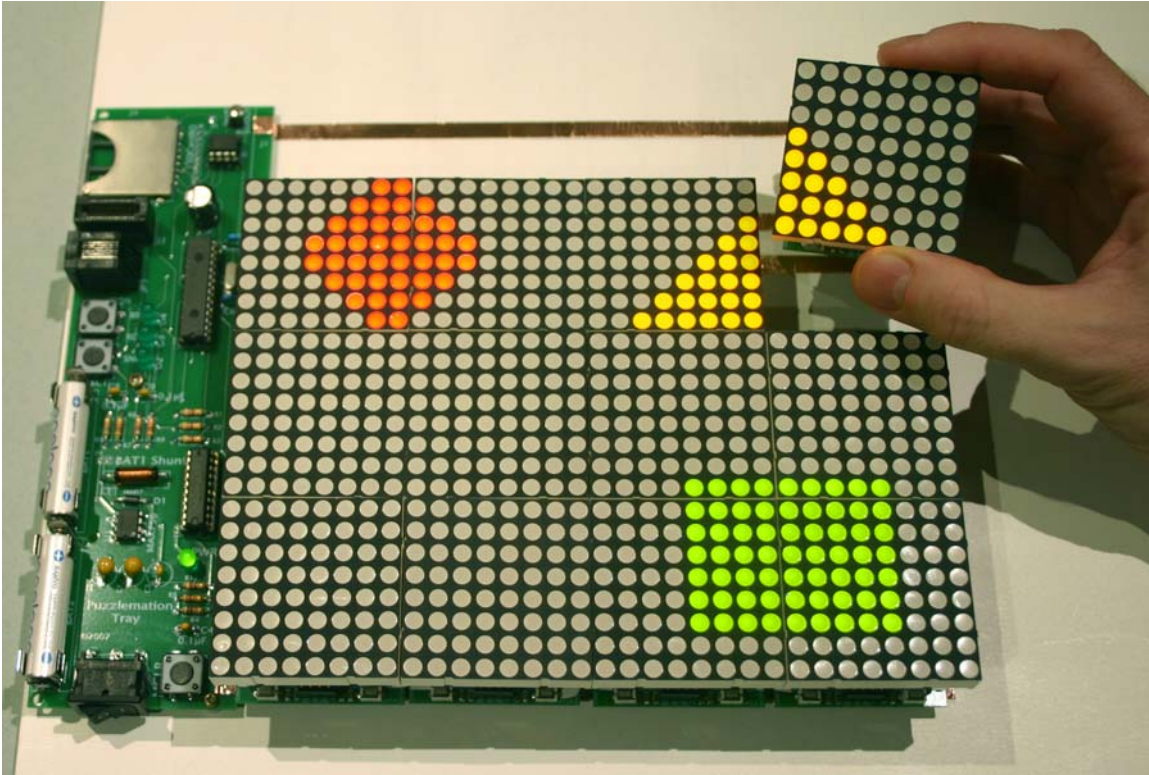


Project MT2287 Puzzlemation - A Dynamic Tile Display

Abstract



Overview

Every time we turn on a computer screen or look at an animated sign, millions of pixels of light disappear into our work, our correspondence, or our entertainment. Imagine if you could capture those dots of light and scoop them up as you would grains of sand on the beach. Now imagine being able to use them as a tool.

Puzzlemation is an expandable platform of light and animation. It can be used to create things as diverse as modular animated signs that can be changed by rearranging its tiles, to a uniquely challenging animated puzzle. First and foremost, it is a tool that lets you create animated light displays of any size and shape for whatever applications you can dream up.

This project's display is made of a number of **tiles**, about 2" square with an 8 x 8 array of color LED pixels. Each tile is *individually* powered and animated, so you can freely pick them up and re-arrange them. To set up a display, the tiles are placed in a special **tray**. Animations are downloaded into the tray via

Ethernet and stored locally on an EEPROM, or loaded via an SD card. The tray broadcasts the animation to each of the tiles, and then synchronizes them.

If the pieces are left in the tray, the animation can be updated continuously over the Ethernet connection. If the tiles are removed from the tray, they'll display the animation for several hours with their own re-chargeable battery power.

Once the animation is synchronized and running on the tiles, you can pick them up and place them anywhere. The display is completely reconfigurable. Need a tall thin display, a square one, or a long skinny one? No problem – re-arrange the tiles as you please.

Technical Description

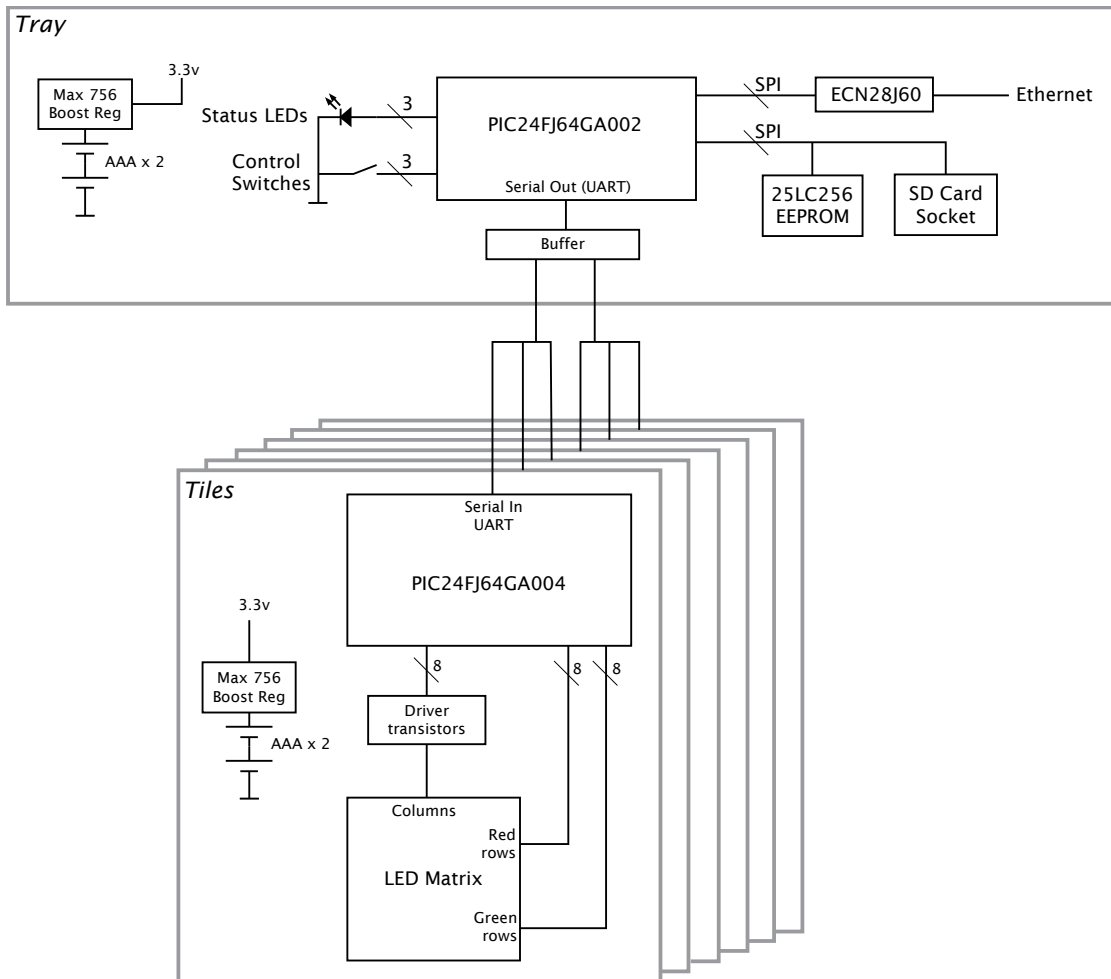


Figure 1 - Block Diagram

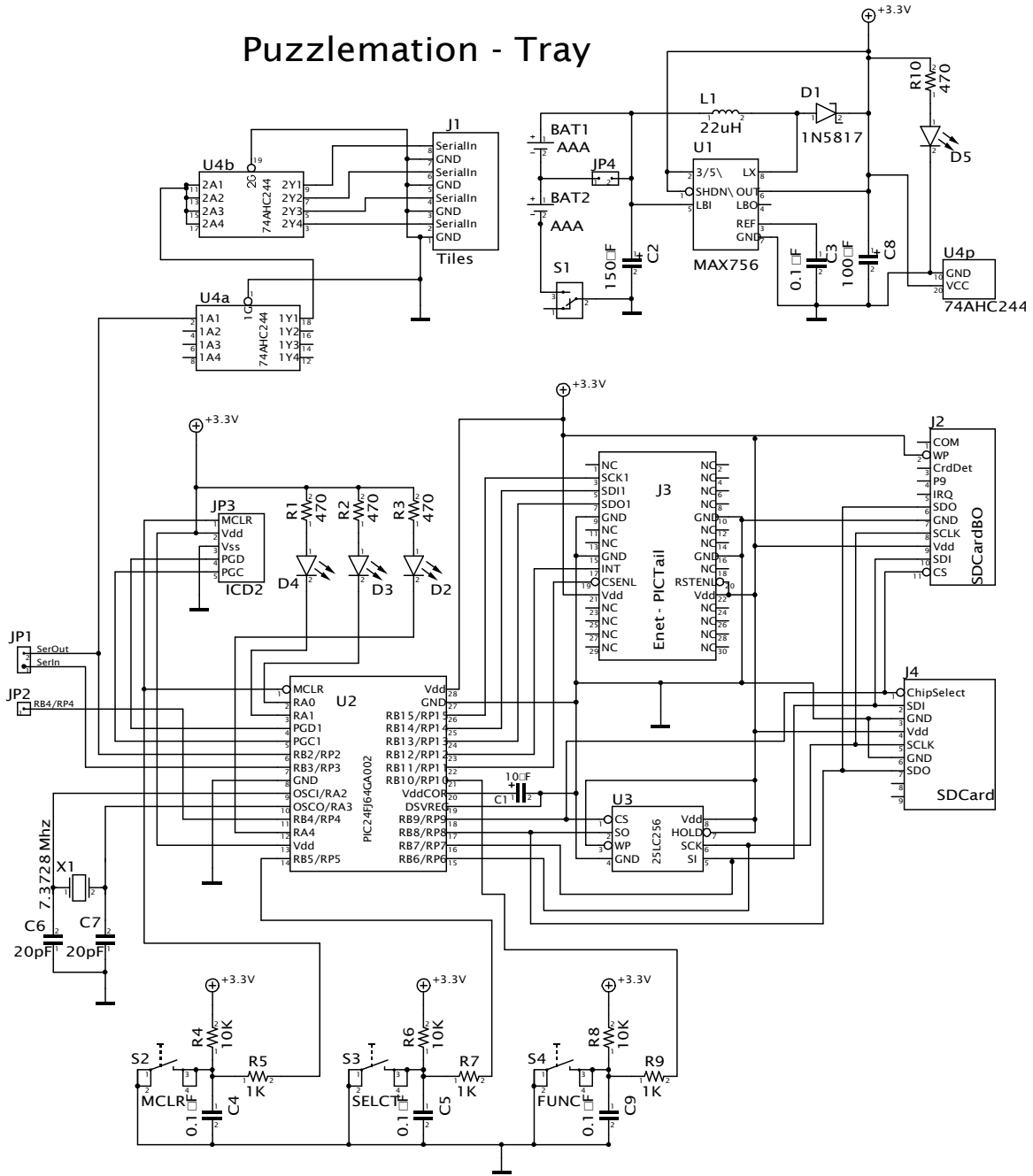
The system has two main components, the tray and the tiles. The tiles are based on 2" square 8x8 LED displays. Underneath each display, a PIC24FJ64GA004

MCU drives the LEDs. Power for each tile is supplied by two AAA batteries, which is fed through a boost regulator to get the 3.3V required. At the bottom of each tile, spring loaded connectors make contact with signal connections in the tray. The tile listens for these signals to download and synchronize animations.

The tray is based on a PIC24FJ64GA002 MCU. This MCU listens for commands from the Ethernet using the ENC28J60 Ethernet controller. Animations downloaded from the net are stored locally on a 25LC256 EEPROM. Or, alternatively, they may be loaded from an SD card, which shares the same SPI interface as the EEPROM. The tray broadcasts animation data and commands to the tiles over a single serial line (UART). Each tile has a unique ID, and listens for its own animation data, ignoring data sent for other tiles. The tray also features some switches and status LEDs to locally control and synchronize the tiles.

Additional software running on a host computer performs higher level operations, such as taking an animated GIF image and breaking it down into individual animations for each tile.

Puzzlemation - Tray



Code Excerpt (Tile.C)

```

// The frames are stored in rows, but the circuit
// scans in columns. This does the swap.
void rotatePlane( unsigned char * src, unsigned char * dst )
{
    int i, j, pixel;

    for ( i = 0; i < 8; ++i)
        dst[i] = 0;

    for ( i = 0; i < 8; ++i)
        for ( j = 0; j < 8; ++j)           // Unrolling candidate
        {
            pixel = (src[i] & (1<<j)) != 0 ? 1 : 0;
            dst[j] |= (pixel << i);
        }
}

void setFrame( AnimationFrame * frame )
{
    unsigned char * framePtr = frame->frameData;

    rotatePlane( &framePtr[RED_OFF], &displayBuffer[RED_OFF] ); // Reds
    rotatePlane( &framePtr[GRN_OFF], &displayBuffer[GRN_OFF] ); // Greens

    // Reset the frame timer
    TMR2 = 0;           // Clear timer 2
    PR2 = frame->frameTime;
    T2CON = 0x8030;    // Turn timer on, 256:1 prescale
}

void setAnimation( AnimationFrame * frames, unsigned int frameCount )
{
    currentAnimation = frames;
    numFrames = frameCount;
    currentFrame = 0;
    setFrame( &currentAnimation[currentFrame] );
}

int main(void)
{
    //
    // Basic initialization
    //
    CLKDIV = 0;

    AD1PCFG = 0xFF;           // Shut off all the analog stuff

    RPINR18bits.U1RXR = 2;    // Make Pin RP2 U1RX
    RPOR1bits.RP3R = 3;      // Make Pin RP3 U1TX

    PMCONbits.PMPEN = 0;     // Disable parallel port
    PMAEN = 0;

    //
    // Set up I/O ports
    //
    TRISA = 0x006F;          // A0-3 input, A4=LG0, A7-A10=LR4..7
    TRISB = 0x000F;          // B0,1=dbg, B2,3=SerIO,
    // B4-B7=LR0..3, B8-B15=LC0..7
    TRISC = 0x0007;          // RC0-3 input, C3-9=LG1..9

    //
    // Timer 1 is the scanline timer
    //
    TMR1 = 0;                // Clear timer 1
    PR1 = (FCY / 1) / 1000;  // Divide by prescale to get clks/sec,
    // divide by 1000 to get clks / lms
    _T1IF = 0;              // Clear interrupt flag
}

```

```

    _T1IE = 1;           // Set interrupt enable bit
    _T1IP = 3;          // Timer lower interrupt priority
    T1CON = 0x8000;     // Fosc/4, 1:1 prescale, start TMR1

    //
    // Timer 2 is the frame timer
    //
    TMR2 = 0;           // Clear timer 2
    PR2 = (FCY / 256);
    _T2IF = 0;         // Clear interrupt flag
    _T2IE = 1;         // Enable T2 interrupt
    _T2IP = 4;         // Higher priority than scanline
    T2CON = 0x8030;    // Turn timer on, 256:1 prescale

    //
    // Change pin - for pushbutton
    // CN8 is on RC0 (pin25)
    //
    _CN10IE = 1;       // Enable interrupt from CN pin
    _CN10PUE = 0;      // Turn off pull-up (already have an external one)
    _CNIF = 0;         // pin Change Notification - Clear interrupt flag
    _CNIE = 1;         // enable Change Notification interrupt
    _CNIP = 2;         // Lowest priority

    //
    // Initialize the UART
    //
    _CN6PUE = 1;       // Enable pull-up to avoid junk in serial in
    U1BRG = BRGVAL;
    U1MODE = 0x8000;   // Reset UART to 8-n-1, alt pins, and enable
    U1STA = 0x0040;    // Reset status register and enable RX

    _UIRXIF = 0;       // Clear UART RX Interrupt Flag
    _UIRXIP = 5;       // Higher priority.
    _UIRXIE = 1;       // Enable interrupts

    //
    // Set up the address of the tile
    //
    _CN8PUE = 1;       // Turn on the pull ups for a zinkosecond. The straps
    _CN9PUE = 1;       // pull to ground, and this makes sure they read right.
    _CN2PUE = 1;
    _CN3PUE = 1;

    tileID = (_RA0 << 3) | (_RA1 << 2) | (_RC0 << 1) | _RC1;
    tileID ^= 0x0F;    // Flip the bottom four bits (jumpers pull to 0)

    _CN8PUE = 0;       // Microchip tech support is squeamish about this ("draws
    _CN9PUE = 0;       // too much current") so we don't leave them on.
    _CN2PUE = 0;
    _CN3PUE = 0;

    setFrame( &currentAnimation[0] );

    // Just hang out and wait for interrupts
    while (1)
        Idle();

    return 0;
}

void displayCol( int colNum, unsigned char redData, unsigned char grnData )
{
    ColStruct redCol, grnCol;
    PortAStruct shadowA;
    PortBStruct shadowB;
    PortCStruct shadowC;

    shadowA.portA = 0;
    shadowB.portB = 0;

```

```

    shadowC.portC = 0;

    redCol.colBits = redData;
    grnCol.colBits = grnData;

    shadowB.columnSelect = (1<<colNum); // Will be inverted by the transistor array

    shadowA.LEDred4_7 = redCol.redBits4_7;
    shadowB.LEDred0_3 = redCol.redBits0_3;

    shadowA.LEDgrn0 = grnCol.grnBits0;
    shadowC.LEDgrn1_7 = grnCol.grnBits1_7;

    LATA = shadowA.portA;
    LATB = shadowB.portB;
    LATC = shadowC.portC;
    PORTC = shadowC.portC;
}

// Column scan timer
void __attribute__((interrupt, no_auto_psv)) _T1Interrupt(void)
{
    IFS0bits.T1IF = 0; // clear interrupt flag

    currentCol = (currentCol + 1) & 0x7;
    displayCol( currentCol,
                displayBuffer[currentCol + RED_OFF],
                displayBuffer[currentCol + GRN_OFF] );
}

// Frame increment timer
void __attribute__((interrupt, no_auto_psv)) _T2Interrupt(void)
{
    _T2IF = 0; // clear interrupt flag

    currentFrame++;
    if (currentFrame >= numFrames)
        currentFrame = 0;
    setFrame( &currentAnimation[currentFrame] );

    // Frame time of zero means freeze (static picture)
    if (currentAnimation[currentFrame].frameTime == 0)
        T2CONbits.TON = 0;
}

// Pushbutton interrupt (currently halts animation)
void __attribute__((interrupt, no_auto_psv)) _CNInterrupt(void)
{
    IFS1bits.CNIF = 0; // pin Change Notification - Clear interrupt flag
    if (PORTCbits.RC2 == 1)
    {
        T2CONbits.TON = ~T2CONbits.TON;
    }
}

void startTestMode(void)
{
    int i;

    T2CONbits.TON = 0;
    for (i = 0; i < 8; ++i)
    {
        displayBuffer[i + RED_OFF] = 0xFF;
        displayBuffer[i + GRN_OFF] = 0x00;
    }
}

// Serial receive. All command processing happens here.
void __attribute__((interrupt, no_auto_psv)) _UIRXInterrupt(void)
{
    static unsigned int commandState = IDLESTATE;

```

```

static unsigned int numIncomingFrames = 0;
static unsigned int numDownloadedFrames = 0;
static unsigned int incomingFrame = 0;
static int incomingFrameByte = 0;
static unsigned int testCount = 0;

unsigned char data;

_U1RXIF = 0; // Clear interrupt flag
data = (unsigned char) U1RXREG;

switch (commandState) {
case IDLESTATE:
    switch (data) {

    case kResetCommand:
        setAnimation( startupFrames, NUMFRAMES( startupFrames ) );
        T2CONbits.TON = 1; // Just in case we were frozen
        break;

    case kStartCommand:
        if (numDownloadedFrames == 0)
            setAnimation( startupFrames, NUMFRAMES( startupFrames ) );
        else
            setAnimation( displayFrames, numDownloadedFrames );
        break;

    case kDownloadCommand:
        commandState = DOWNLOADSIZESTATE;
        break;

    case kFreezeCommand:
        T2CONbits.TON = 0;
        break;

    case kUnfreezeCommand:
        T2CONbits.TON = 1;
        break;

    case kIdentifyCommand:
        setAnimation( &tileIDFrames[tileID], 1 );
        break;

    case kTestCommand:
        testCount = 0;
        startTestMode();
        commandState = TESTSTATE;
        break;

    case kTestOneCommand:
        testCount = 0;
        commandState = TESTONESTATEID;
        break;

    case kVerifyCommand:
        setAnimation( verifyFrames, NUMFRAMES( verifyFrames ) );
        break;

    default:
        break;
    };
    break;

case TESTONESTATEID:
    if (data == tileID)
    {
        startTestMode();
        commandState = TESTSTATE;
    }
    else
        commandState = TESTIGNORESTATE;

```

```

        break;

    case TESTSTATE:
        displayBuffer[testCount + RED_OFF] = 0x00;
        displayBuffer[testCount + GRN_OFF] = data;
        // Fall thru...

    case TESTIGNORESTATE:
        testCount++;
        if (testCount == 8)
            commandState = IDLESTATE;
        break;

    case DOWNLOADSELECTSTATE:
        if (data != tileID)
        {
            commandState = DOWNLOADIGNORESTATE;
        }
        else
        {
            commandState = DOWNLOADSTATE;
            setAnimation( downloadFrames, NUMFRAMES( downloadFrames ) );
        }
        break;

    case DOWNLOADSIZESTATE:
        numIncomingFrames = (unsigned int) data;
        incomingFrame = 0;
        incomingFrameByte = -2;
        commandState = DOWNLOADSELECTSTATE;
        break;

    case DOWNLOADIGNORESTATE:
    case DOWNLOADSTATE:
        if (incomingFrame < kMaxDisplayFrames)
        {
            if (commandState == DOWNLOADSTATE)
            {
                if (incomingFrameByte == -2)
                    displayFrames[incomingFrame].frameTime
                        = ((unsigned int)data) << 8;
                else if (incomingFrameByte == -1)
                    displayFrames[incomingFrame].frameTime
                        |= (unsigned int)data;
                else
                    displayFrames[incomingFrame].frameData[incomingFrameByte] = data;
            }
            incomingFrameByte++;
            if (incomingFrameByte == 16)
            {
                incomingFrameByte = -2;
                incomingFrame++;
            }
        }
        if (incomingFrame == numIncomingFrames)
        {
            if (commandState == DOWNLOADSTATE)
            {
                setAnimation( waitingFrames, NUMFRAMES( waitingFrames ) );
                numDownloadedFrames = numIncomingFrames;
            }
            commandState = IDLESTATE;
        }
        break;
    }
}

```