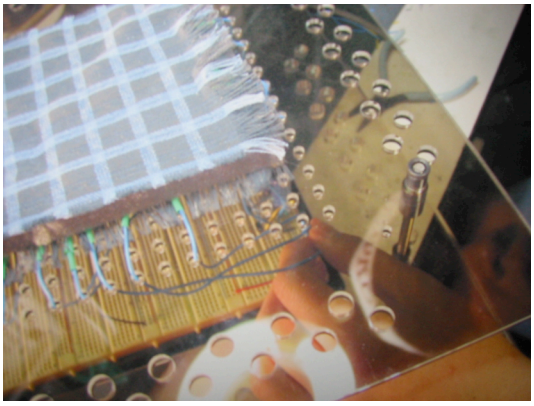
TechStyle System: The Management and Memory of Electronic Fabric Input and Output

MIT Undergraduate Thesis



Abraham Hsu (madentai@mit.edu)

Lab:	MIT Media, Arts, Sciences, Media Laboratory 20 Ames Street Cambridge, MA 02139 In collaboration with *Collins & Aikman*
Group:	Media Fabrics Group, TechStyle
Supervisors:	Glorianna Davenport (gid@media.mit.edu) Hyun-Yeul Lee (hyun@media.mit.edu)
Alumni Contributors:	Alex Crumlin and Charles Hightower
Date:	May 10th, 2004

Abstract

"The two main objectives are management and memory. At any time, the e-fabric unit has two states: pixel input and pixel output. The input state of a unit describes whether or not each pixel has been pressed down. The output sate of a unit describes whether or not each pixel has had a color change (on/off). Management is the ability to change and control both input and output states over time. Memory is the ability to record the history of both input and output states over time. The management and memory of a single e-fabric unit can be combined with other Units to create a powerful collective system of pixel input and pixel output states over time."

Table of Contents

List of Figures

i. Circuit View of the Electronic Fabric

The electronic fabric was custom-designed by Hyun-Yeul Lee. It is made with cotton, nylon, and conductive fiber. This document assumes a certain amount of knowledge of the e-fabric. Please read Appendix A: Electronic Fabric with Simultaneous Input and Output for a review of the e-fabric implementation details. Also note that the terms e-fabric and fabric are used synonymously in this document.

An e-fabric *'unit'* is defined as an 8x8 pixel square of e-fabric. A unit is seen as the optimal integration of the fabric with the associated circuit board. A square 'X' by 'X' pixel fabric retains the best fabric area to number of wires ratio. At a maximum of 34 I/O pins on a PIC microcontroller, an 8x8 pixel was thought to be the largest fabric unit possible (8*4 lines). As an extra positive, the 8x8 pixel unit allows for easier PIC coding because hardware memory is usually organized in bytes and 1 byte is 8 bits.

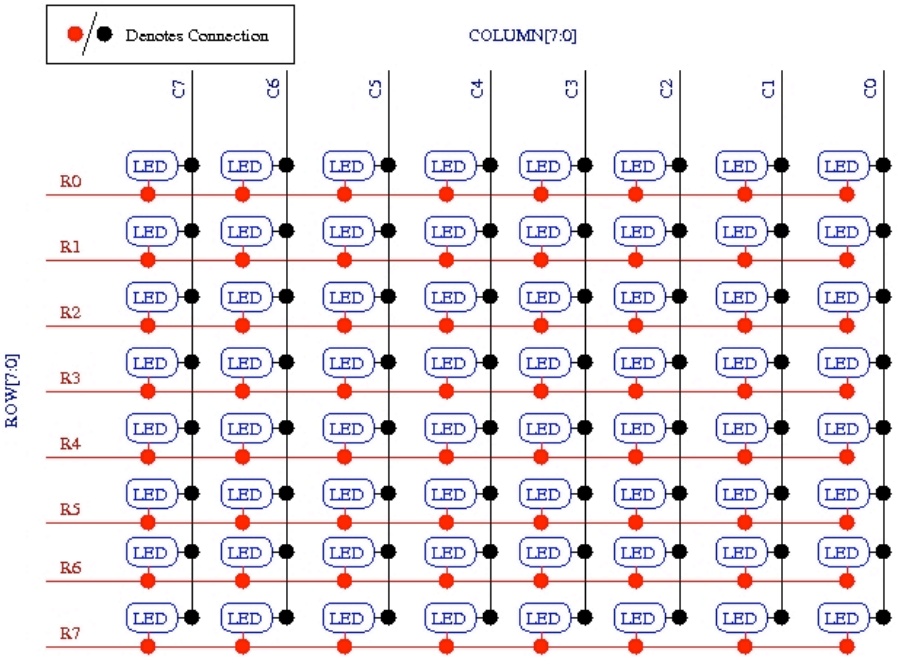


Figure 1: TechStyle LED Fabric Schematic

The unit's output network can be idealized as a circuit of 64 LEDs and wires (Figure 13). The unit's input network can be idealized as the same schematic of Figure 13, except all the LEDs are shorted as simply connected wires. These two ideal input and output circuits can be constructed to test the TechStyle system. The 32 I/O lines of the e-fabric unit are connected to a PIC (see Figure 5).

An example of the unit's output: how to turn 'on' pixel (or LED in the idealized case) in row1, column5? The answer: pass power (high voltage) to column5 and tie ground (low voltage) to row1. Moreover, leave the rest of the column and row lines as unconnected open-circuits (no voltage).

I. Overview

1.1 Purpose and Background

The two main objectives are management and memory. At any time, the e-fabric unit has two states: pixel input and pixel output. The input state of a unit describes whether or not each pixel has been pressed down. The output sate of a unit describes whether or not each pixel has had a color change (on/off). Management is the ability to change and control both input and output states over time. Memory is the ability to record the history of both input and output states over time. The management and memory of a single e-fabric unit can be combined with other Units to create a powerful collective system of pixel input and pixel output states over time.

1.2 System Overview

The TechStyle project can be divided into two main systems: the CPLD System and the Units System (Figure 2).

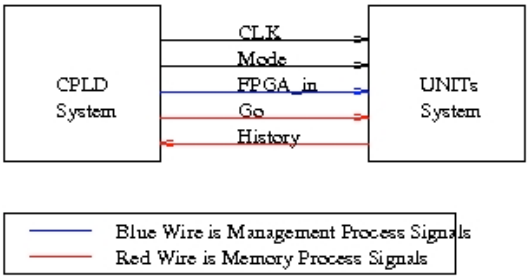


Figure 2: TechStyle System Overview Schematic

1.3 TechStyle System Signals

There are five single-bit signals (Figure 2) that travel between the CPLD System and the Units System. Four are outputs from the CPLD System to the Units system, and one is an input to the CPLD system from the Units system:

- System Clock (CLK): Output produced by the CPLD system. It is a 1 Mhz system clock that synchronizes the system.
- Mode (Mode): Output from the CPLD system. When it is high voltage, the pixel output is in RAMmode. When it is low voltage, the pixel output is in AUTOmode (see 3.4 PIC Pixel Output).
- RAM Picture (FPGA_in): Serial output from the CPLD system. It holds pixel output data used during the RAMmode (see 3.2 CPLD Pixel Output).
- History Trigger (Go): Output from the CPLD system. When it is a high voltage, it is a sign to the Units system to start the output of its History data (see 3.3 CPLD Pixel Input and Computer Input).
- History (History): Serial data input to the CPLD system. Part of the memory procedure, it passes data describing pixel output states of all pixels in the TechStyle system over time(see 3.3 CPLD Pixel Input and Computer Input). (An analogy is that the Units System uses this medium to pass color snapshot pictures of the pixels to the CPLD System over time.)

II. Two Major System Components

2.1 CPLD System

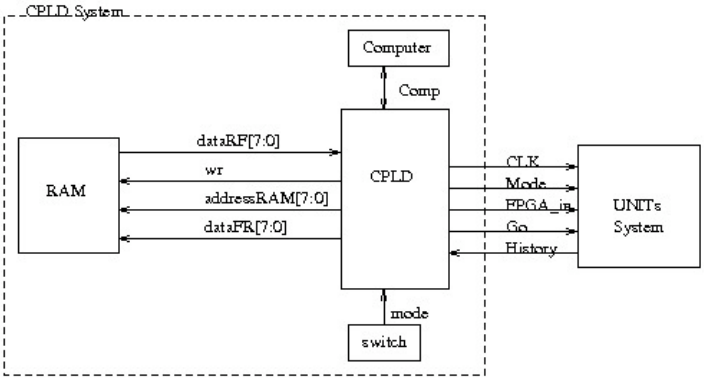


Figure 3: CPLD System Schematic

The CPLD System is composed of four components: CPLD, RAM, switch, and Computer (Figure 3). (Note that the RAM is instantiated within the memory of the CPLD, therefore a physical integrated chip of a RAM is not needed. This allows for nearly instantaneous communication between the RAM and CPLD.) The CPLD is the primary element of the CPLD System, and the following inputs and outputs shall be referenced to the CPLD.

2.1.1 CPLD System Building Blocks

Complex Programmable Logic Devices (CPLD)
Four ATF750C CPLDs were required to fit the VHDL program compiled with Altera's MaxPlusII software. The VHDL code was programmed with the Galaxy software in the MIT Electrical Engineering Lab, 38-600.

Switch
The single switch can be implemented with a variety of simple switches, buttons, latches, etc.

Computer
In the future, the computer may be a part of the management and memory procedures. It receives and sends data on the pixel output states over time.

2.1.2 CPLD Inputs

There are four inputs to the CPLD (Figure 3). One from the Units System, one from the switch, one from the RAM, and one from the Computer:

- History (History): Serial data input to the CPLD. Part of the memory procedure, it passes data describing pixel output states of all pixels in the TechStyle system over time(see 3.3 CPLD Pixel Input and Computer Input).
- Switch (mode): User-controlled input from a physical switch to the CPLD. When it is high voltage, the pixel output is in RAMmode. When it is low voltage, the pixel output is in AUTOmode (see 3.4 PIC Pixel Output).
- RAM Data (dataRF[7:0]): Input from the RAM. It carries the 8-bit data referenced by the address signal.
- Computer (Comp): Serial output from the Computer. It holds pixel output data for every pixel in the TechStyle system. In the future, information may be used during the RAMmode.

2.1.3 CPLD Outputs

There are eight outputs from the CPLD (Figure 3). Four to the Units System, three to the RAM, and one to the Computer:

- System Clock (CLK): Output produced by the CPLD. It is a 1 Mhz system clock that synchronizes the Units System.
- Mode (Mode): Output from the CPLD to the Units System. When it is high voltage, the pixel output is in RAMmode. When it is low voltage, the pixel output is in AUTOmode (see 3.4 PIC Pixel Output).
- RAM Picture (FPGA_in): Serial output from the CPLD to the Units System. It holds pixel output data used during the RAMmode (see 3.2 CPLD Pixel Output).
- History Trigger (Go): Output from the CPLD to the Units System. When it is a high voltage, it is a sign to the Units system to start the output of its History data (see 3.3 CPLD Pixel Input and Computer Input).
- Write (wr): Output from the CPLD to the RAM. When it is a high voltage, it allows the referenced RAM address to be written by the signal dataFR.
- RAM Address (addressRAM[7:0]): Output from the CPLD to the RAM. It is an 8-bit signal input to the RAM which references a unique 8-bit data string in the RAM.
- Write Data (dataFR[7:0]): Output from the CPLD to the RAM. It is an 8-bit signal which is written, when write is high voltage, to the 8-bit data string referenced by the RAM address signal.
- Computer (Comp): Serial input from the Computer. It holds data for every pixel in the TechStyle system. Should be identical to the CPLD Pixel Output signal (see 3.2 CPLD Pixel Output). In the future, information may be used during RAMmode.

2.2 Units System

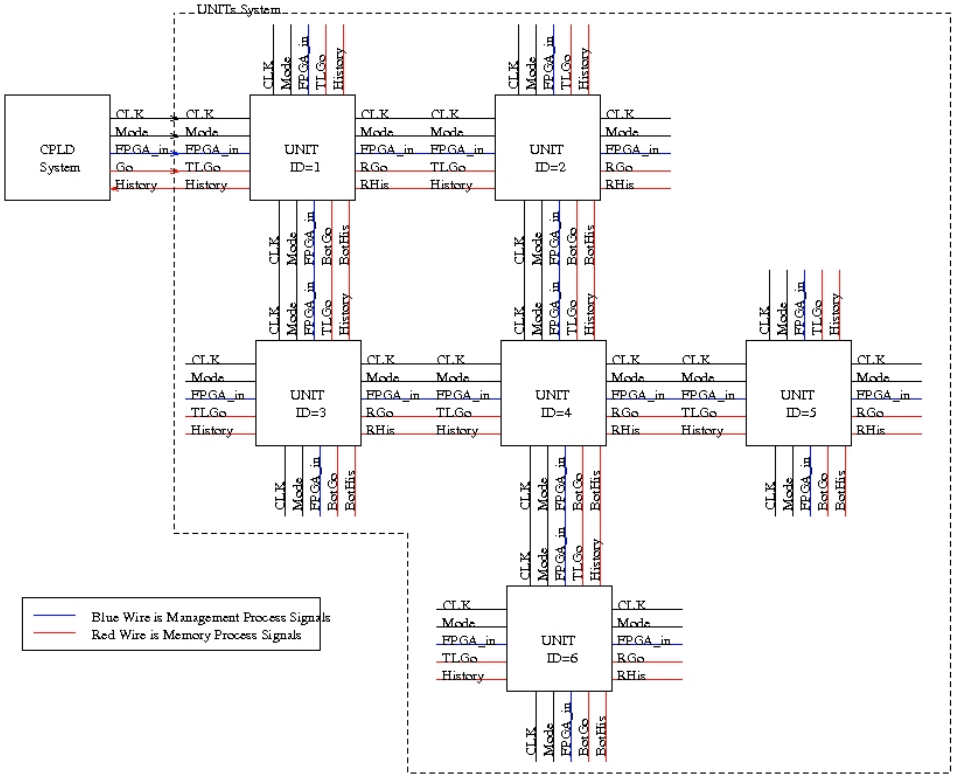


Figure 4: Units System Schematic

The Units System is composed of multiple identical units connected together (Figure 4). This makes the design scalable and flexible. Each unit has five I/O signals connected to its left, right, top, and/or bottom units. The exception is the top-left unit, which connects its five left I/O signals to the CPLD System. A single unit's internal signal schematic is provided in Figure 4. In following, the CLK, Mode, and FPGA_in signals are the same for all units.

Thirty-two signals are connected between the PIC and the fabric unit (Figure 4). (Refer to Figure 1 for ideal fabric unit schematic.) Sixteen MOSFETs are placed between the PIC outputs related to the management process, and the fabric unit. In this MOSFET configuration, a low PIC output seems like an open circuit and a high PIC output seems like a connection to power. The 8 PIC inputs related to the memory process are connected to the 8 column ends of the fabric unit. The 8 PIC outputs related to the memory process are connected to the 8 row lines of the fabric unit.

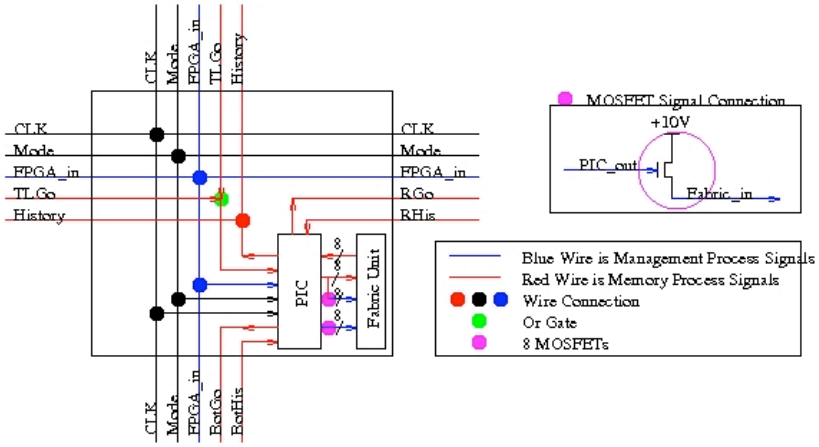


Figure 5: Single Unit Schematic

2.2.1 PIC System Building Blocks

PIC Microcontroller
One PIC16F452 is needed for every unit being created. The language used is Assembly and was compiled with MicroChip's MPLab. The programmer was MicroChip's PIC-StartPlus. (Note that the primary component of the Unit is the PIC, and the following inputs and outputs shall be referenced to the PIC.)

Metal-Oxide-Semiconductor Field-Effect Transistors (MOSFET)
Sixteen 2N7000 MOSFETs per unit are needed for the pixel outputs to the fabric.

OR Gate
A single dual-input 'or gate' per unit is required. At the moment, the 74LS32 quad dual-input 'or gate' is being used.

Electronic Fabric
An 8x8 pixel electronic fabric is required per unit. The electronic fabric, custom designed by Hyun-Yeul Lee (hyun@media.mit.edu), is the cornerstone of the TechStyle System.

2.2.2 PIC Inputs

There are seven inputs to the PIC (Figure 5), including one 8-bit parallel signal from the Fabric:

- System Clock (CLK): Produced by the CPLD system. It is a 1 Mhz system clock that synchronizes the system.
- Mode (Mode): From the CPLD system. When it is high voltage, the pixel output is in RAMmode. When it is low voltage, the pixel output is in AUTOmode (see 3.4 PIC Pixel Output).

- RAM Picture (FPGA_in): From the CPLD system. It holds pixel output data used during the RAMmode (see 3.2 CPLD Pixel Output).
- Top and Left Trigger (TLGo): The *'or'* of the BotGo of the top unit and the RGo of the left unit (BotGo *or* RGo). When its voltage is high, it is a sign to start the output of its History data (see 3.5 PIC Pixel Input (History)). The 'or gate' conserves the limited number of PIC pins used.
- Right History (RHis): Serial data input to the PIC. Part of the memory procedure, it passes data describing pixel output states of all pixels to the right and bottom of the right unit (see 3.5 PIC Pixel Input (History)).
- Bottom History (BotHis): Serial data input to the PIC. Part of the memory procedure, it passes data describing pixel output states of all pixels to the right and bottom of the bottom unit (see 3.5 PIC Pixel Input (History)).
- Fabric Input [7:0]: Eight parallel inputs from the fabric which is used to determine the state of all 64 pixels (see 3.4 PIC Pixel Output).

2.2.3 PIC Outputs

There are five outputs from the PIC (Figure 5), including three 8-bit parallel signals from the fabric unit:

- Right Trigger (RGo): When its voltage is high, it is a sign to the right unit to start the output of its Right History (see 3.5 PIC Pixel Input (History)).
- Bottom Trigger (BotGo): When its voltage is high, it is a sign to the bottom unit to start the output of its Bottom History (see 3.5 PIC Pixel Input (History)).
- Fabric Input and Output [7:0]: Eight parallel outputs from the PIC used for both checking *and* changing the state of the fabric. The signals for checking the state of the fabric is output directly to the fabric. The signals for changing the state of the fabric are passed through eight MOSFETs before being connected to the fabric.
- Fabric Output Only [7:0]: Eight parallel outputs from the PIC used to change the state of the fabric. These signals are passed through eight MOSFETs before being connected to the fabric.

III. System Finite State Machine (FSM) Components

3.1 RAM

The RAM is used as a storage device of a single state of all the Units of the TechStyle system. It essentially holds a single snapshot 'picture' of all the united combined. The RAM has an 8-bit address with 8 bits (8bits=1byte) referenced per address. The memory organization (Figure 6) is set at nine bytes per unit. Within each unit's nine bytes, the first byte is the unit's unique id. The next eight bytes describe the 64 pixels of the unit. Each byte of these eight bytes represents a row, starting from Row0, to Row1, etc., to Row7. The 8-bits within each byte represent the columns 7 down to 0. A 1 means the pixel is on, while a 0 means the pixel is off.

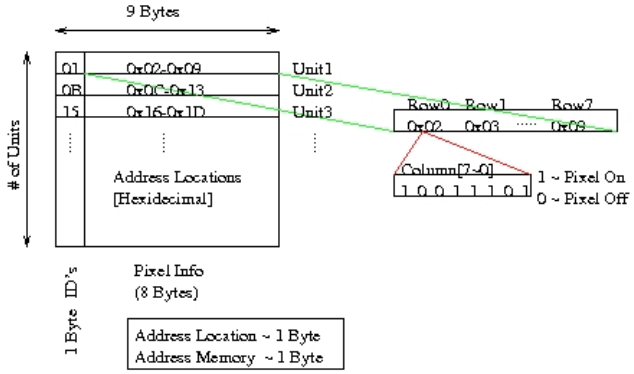


Figure 6: RAM Memory Organization

3.2 CPLD Pixel Output

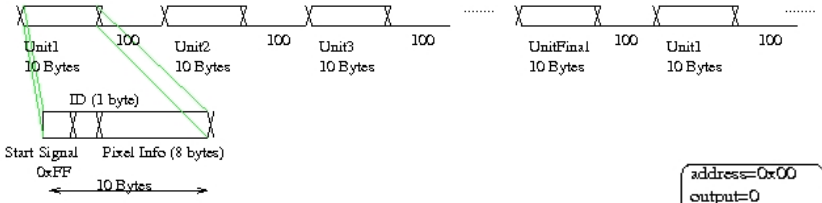


Figure 7: CPLD Serial Output Timing Diagram

(Note that the signal referred to as 'output' in this section is FPGA_in.) The FPGA output has a specified format shown in Figure 6. The FPGA output is taken directly from the RAM data. The first unit's information at address 0x00 listed in the RAM memory is the initial output (Figure 7). Then there is a 100-bit pause low before the second unit's information is output. This pattern is repeated until all the Units in the TechStyle system have been outputted, and then this whole cycle repeats from the first unit in the RAM. Each unit's information consisted of a start byte of 0xFF, then its 1 byte id, and finally 8 bytes of its pixel information.

The CPLD serial output process (Figure 8) was implemented as a finite state machine with six states: Initial, Setup, Start, Byte7, Byte, and Pause1. This process continues whether or not the mode is high or low voltage, though the PICs only process the information only when mode is in RAMmode (mode='1').

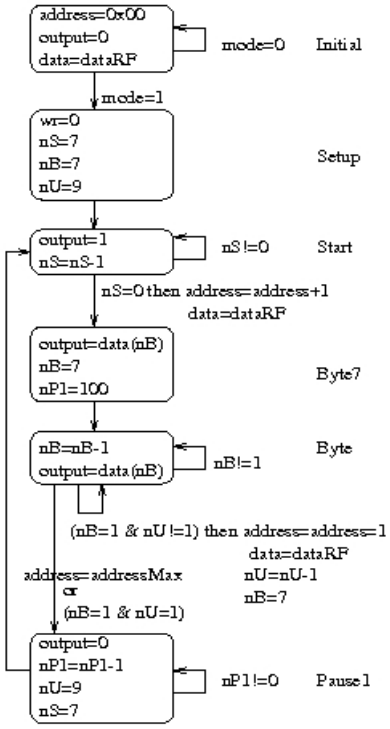


Figure 8: CPLD Serial Output Process

3.3 CPLD Pixel Input and Computer Input

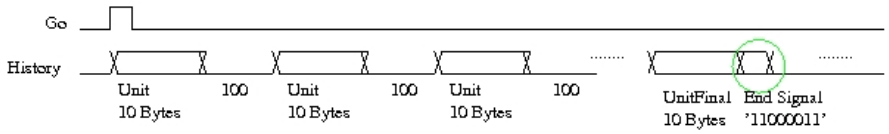


Figure 9: CPLD Serial Input Timing Diagram

The CPLD input is similar to the CPLD output, but there are many important differences (Figure 9). The CPLD input's 10 bytes of unit data is identical in format to that of the output. Also, the unit data are separated by 100 bits of low. The similarity ends there.

Knowing the differences are important to understanding the more complex CPLD input data stream. The input has an End signal of '11000011' (0xC3) connected to the end of the final unit input. Also, the CPLD input does not automatically continue to repeat itself. When the CPLD output signal Go is voltage high, it is the sign which allows the History to start passing its data.

The most important detail is that the CPLD input does not follow the same sequence of Units as the 3.2 CPLD Output. While the 3.2 CPLD Output followed the order in the RAM, the CPLD input has a much more complicated, but still organized, order (see 3.5 PIC Output (History)). In the future, the CPLD Input may be passed directly as an output to the Computer. This sequence of units's data will be interpreted by the Computer.

3.4 PIC Pixel Output

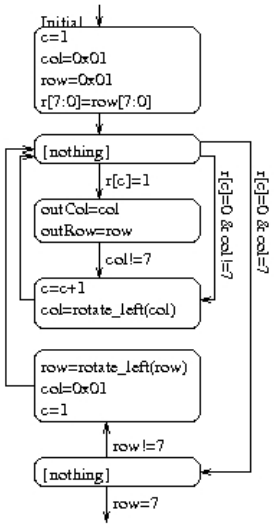


Figure 10: PIC Pixel Output

The PIC holds the 64-pixel states in 8 addresses: row0, row1, etc., to row7. Each address contains a byte of information which signifies columns 7 down to 0. This format is similar to that of the RAM Memory Organization (see 3.1 RAM). An output process was created to interpret all the states of the pixels (Figure 10). This process initializes at row0, column 0. It cycles through every column of row0, from 0 to 7, checking if the pixel is on ('1') or off ('0'). After this, row1 is checked in the same manner, and then row2, row3, etc.

If the column in a row is determined to be on ('1'), then the PIC outputs signals, passing power through its ports to the specified pixel (see i. Electronic Fabric).

3.4.1 RAMmode

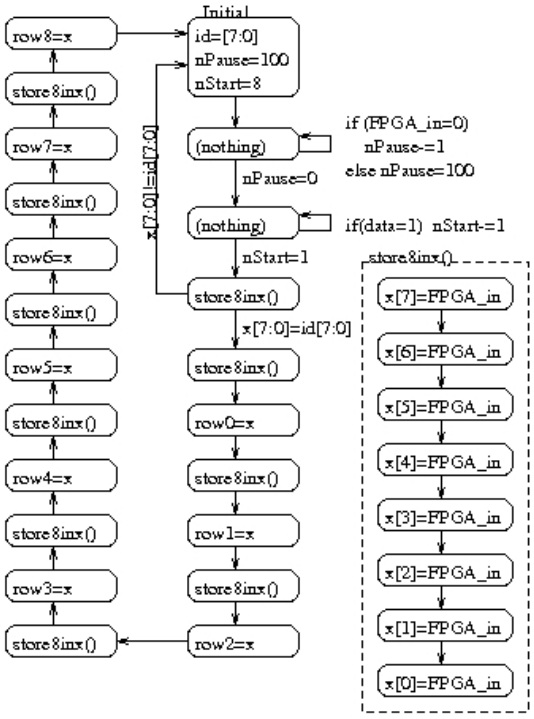


Figure 11: RAMmode Pixel Output Process

The RAMmode output draws the pixel picture specified in the RAM. It takes in the serial input from the CPLD (see CPLD Pixel Output) and interprets this data into pixel output states (Figure 11). After the initial state, the process waits for at least 100 low voltage inputs from FPGA_in before moving to the next state. Then it waits for a Start byte of 8 high voltage inputs from FPGA_in. The next 8 bits are stored in x[7:0] by function store8inx(). If x is not equal to the id of the specific unit, then it moves back to state initial. Otherwise, it moves on to repeating a pattern of storing 8 bits into x, and then storing this byte into all eight rows. This process is executed only when the mode is voltage high.

3.4.2 AUTOmode

The AUTOmode of pixel output is initiated when the mode signal is low voltage. In this state, the user controls which pixels turn on or off by pressing down on the fabric. The overall logic is simple at every pixel press: 1) If pixel was off, then turn it on. 2) If pixel was on, then turn it off. Every pixel is analogous to a light switch.

3.5 PIC Pixel Input (History)

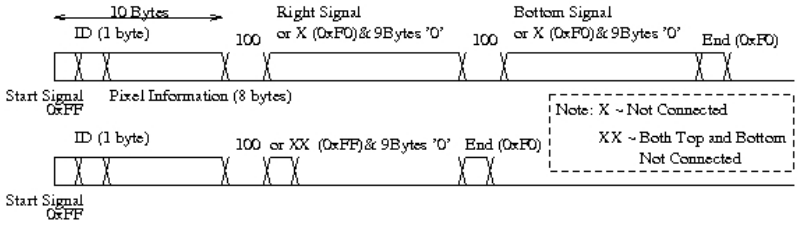


Figure 12: PIC Pixel History Timing Diagram

The PIC Pixel History data starts to pass only when the signal TLGo is high voltage. The History signal follows a general guideline:

1. Start Signal 0xFF
2. Own Unit's 8-bit ID
3. Own Unit's 64-bit Pixel Information
4. Right Signal, or if X ~0xF0, or if XX ~0xFF
5. (If not XX) Bottom Signal, or if X ~ 0xF0

This pattern can be seen in the PIC Pixel History Timing Diagram (Figure 12). In the diagram, the bottom signal represents the signal format if there were no Units connected to the top and bottom (XX). The top signal shows the format of the history signal if a unit was connected to either the top or bottom of the current unit.

An important detail in the history signal is that the 'right signal' is not limited to only 10 bytes, the standard size of one unit. The 'right signal' contains all the pixel data of all the units to its right and bottom. Therefore, it most likely has many signals within it. The same principle applies to the 'bottom signal'.

The PIC Pixel History Process (Figure 13) follows the general guideline of the Present Unit, Right Signal, and then Bottom Signal order mentioned previously.

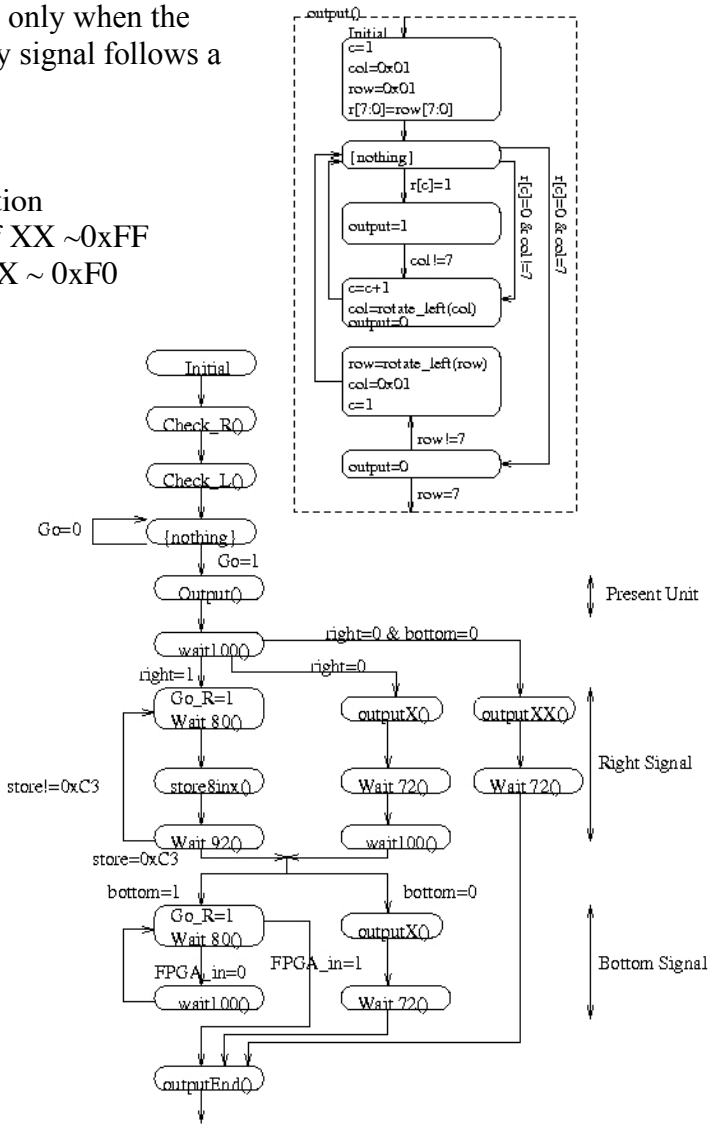


Figure 13: PIC Pixel History Process

IV. Challenges

5.1 Challenges

The fabric was the largest challenge because of its multiple unknowns. The conductive fiber has variable resistance which varies the speed at which pixels turn on. This makes it difficult during implementation of the pixel output, when individual pixels do not visually turn on as a group consistently. A phenomenon termed 'pixel leakage' is a problem that may be due to too much power. This word describes how turning on one pixel, may slight change the color of its surrounding pixels.

Another large challenge was in increasing the power output to the fabric. The maximum voltage allowed to drive the PIC is 7.5 V, but the power wanted for fabric output is 10-15V. The higher amount of voltage increases the speed that the pixel turns on. Ideally, one wants increased speed to decrease, or possibly nullify, the variance of resistance within the fabric.

The first method tried was a MOSFET with the PIC output tied to the gate, and the fabric line tied to the base. The MOSFETs of the column have power tied to the drain, and the MOSFETs of the row have ground tied to the drain. Basically, the key idea was to set power at 10V. When the PIC signal is low, there will be an open circuit, and when the PIC signal is high, power for the column, or ground for the rows, will be passed through the circuit. Unfortunately, the MOSFET output is limited by the gate voltage. For example, when one passes a 5V PIC signal to the gate, the MOSFET output is still ~3.5V even when the drain is tied to 10V.

The second method was utilizing op-amps to increase the voltage output to the fabric. The open circuit voltage was increased to over 10V, but the output drops to ~3V when the op-amp output is connected to the fabric. This may be due to an output resistance that changes the voltage and current.

The third method involves focusing upon increasing the current, instead of the voltage, through BJTs. A simple common-emitter amplifier configuration should be enough to greatly increase the current, and therefore the power. This plan has will be tested on May 14th.

V. Conclusion

The memory portion of the TechStyle System works as expected. It monitors the history of the fabric, and sends data of the pixel output states in a flexible manner.

The management portion related to the input states of the TechStyle system works as expected. Therefore, the sense of whether a pixel has been pressed within all units is known exactly.

The management portion related to the output states of the TechStyle system does not always work as expected. The change of pixel color has not been faultless. The memory of the system has proved to be more resilient. This is mainly due to the challenges of the electronic fabric (see 5.1 Challenges). While the digital signals passed to the fabric units seem to be as planned,

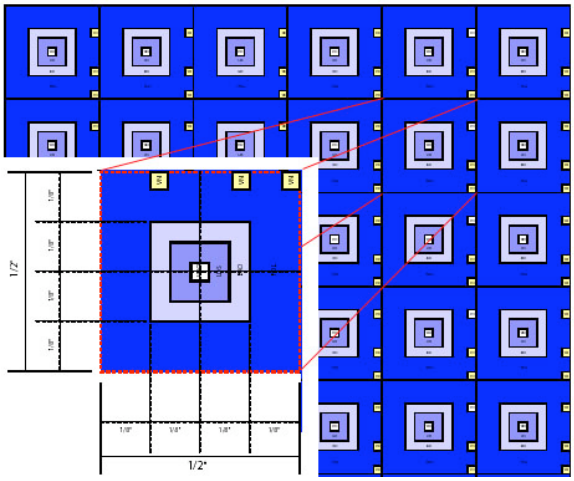
the electronic fabric does not always respond with changes in color of an expected manner. The TechStyle is expected to work with an idealized schematic such as in Figure 1, but the reality of electronic fabric is not ideal.

Further work and communication with the Computer is expected in the future, as occasionally mentioned throughout this document. Eventually, the management and memory of the TechStyle system would be controlled by a user interface application within the computer.

Appendix A: Electronic Fabric with Simultaneous Input and Output

Implementation

Scalability



The scalability issue is approached by using a pixel layout like a Cartesian coordinate system. The pixel layout is similar to the implementation used by a normal television and computer screen. The picture to the right is an example of the current pixel system being researched.

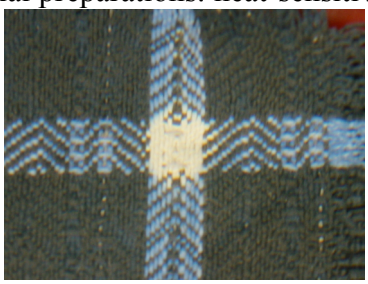
In the picture, each dark blue square represents one pixel and each pixel is a half inch by a half inch. Within each dark blue square is a one-eighth by one-eighth light blue square which represents the output square. There are three input points within each dark blue square pixel and each one is represented as a small yellow square. To improve robustness, three inputs per pixel were chosen in the case that if one input square failed, two other points would still be operational.

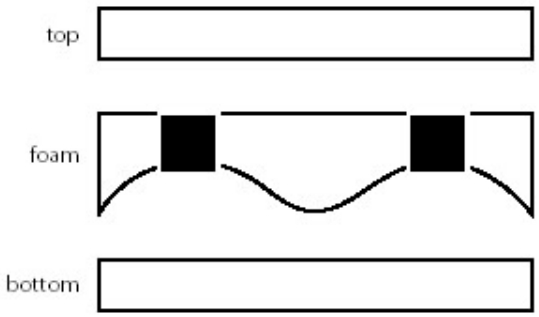
This pixel design is ideal because it can be hypothetically expanded infinitesimally. Also, in viewing this ‘x number’ by ‘y number’ pixel design as an ‘x number’ by ‘y number’ Cartesian coordinate system, it is easy to manage the output and input signals from a higher level thought process.

Simultaneous Input and Output

The output is a change of color within the output square. In the pictures to the right, the overlap of the blue lines represents the output square that is identical within each pixel. The top picture to the right is an example of an output square that is inactivated. The bottom picture is an example of an output square that is activated.

The underlying idea for the output requires two basic material preparations: heat-sensitive dye and wires woven into the fabric. The electronic fabric used by this project has by streaked with custom designed heat-sensitive dye. At this time, the experimental fabric has been coated with dye that is blue at room temperature and white at a higher temperature. Wires are woven within the fabric and current is passed through these wires to induce heat dissipation. This heat causes an increase in temperature at a pixel point and leads to a change in color at an output pixel.





The input senses whether the fabric has been pressed down upon. The input necessitates two layers of woven fabric and a foam layer, as shown in the picture to the left. As the top fabric layer is pressed down, its woven wires come in contact with the bottom layer's woven wires. The current sent through the top wires flows into the bottom wires. This is then detected by integrated circuits connected at the ends of the bottom wires.

ICs Separate from Fabric

The integrated circuits used during this project were not woven underneath or on the circuit. The integrated circuits are placed within a circuit board that can be hidden anywhere within the object the electronic fabric is covering.

Appendix B: VHDL [CPLD] Code

```
-- FRsystem.vhd
-- Abraham Hsu
-- April, 2005
-- Student, MIT

-- Program Description:
-- This program interacts with a ROM and the mother PIC unit. It outputs
the
-- information from the ROM as a serial data on one line to the mother
unit.
--
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
```

```
use IEEE.std_logic_unsigned.all;    -- Needed for "+" with
std_logic_vectors
library work;
--use work.new_func.all;

entity FRsystem is
      port(
            clk               : in std_logic;
            mode        : in std_logic;

            -- ROM signals
            dataRF            : in std_logic_vector(7 downto 0);
            wr          : out std_logic;
            addressRAM      : out std_logic_vector(7 downto 0);
            dataFR          : out std_logic_vector(7 downto 0);

            -- PIC signal
            output          : out std_logic
      );
end entity FRsystem;

architecture component_a of FRsystem is

-- Variables
signal address, data    : std_logic_vector(7 downto 0);

-- States for FRSystemOutput FSM
type StateType is (Initial, Setup, Start, Byte7, Byte, Pause1);
signal  S_s : StateType;

begin

-- Internal Signals
addressMax <= "01010001"; -- addressMax=81
addressRAM   <= address;
dataFR       <= "00000000";


      FRoutput_process : process(clk)
            variable nS : integer range 0 to 7 :=0;
            variable nB : integer range 0 to 7 :=0;
            variable nU : integer range 0 to 9 :=0;
            variable nP1 : integer range 0 to 100 :=0;
      begin
            if rising_edge(clk) then
                  case S_s is
                -- Initial
                        when Initial=> if ((mode='1') and (power='1'))
then
                                          S_s <= Setup;
                                    end if;
                        -- Setup
                        when Setup  => S_s <= Start;
                        -- Start
                        when Start  => if (nS=0) then S_s <= Byte7;
                                                address <= address+1;
                                                data <= dataRF;
```

```
                                    else S_s <= Start;
                                    end if;
                        -- Byte7
                        when Byte7 => S_s <= Byte;
                        -- Byte
                        when Byte   => if (address=addressMax)  then
                                          S_s <= Pause1;
                                          address <= "00000000";
                                       elsif not (nB=1) then
                                          S_s <= Byte;
                                       elsif not (nU=1) then
                                          address <= address+1;
                                          data  <= dataRF;
                                          nU := nU-1;
                                          nB := 7;
                                          S_s <= Byte;
                                        else
                                          S_s <= Pause1;
                                       end if;
                              -- Pause1
                              when Pause1     => if (nP1=0) then
                                                    S_s <= Start;
                                                 end if;
                        when others => S_s <= Initial;
                  end case;

                  -- Initial
                  if S_s=Initial then
                        address <= "00000000";
                        output <= '0';
                        data <= dataRF;
                  -- Setup
                  elsif S_s=Setup then
                        wr <= '0';
                        nS := 7;
                        nB := 7;
                        nU := 9;
                  -- Start
                  elsif S_s=Start then
                        output <= '1';
                        nS := nS-1;
                  elsif S_s=Byte7 then
                        output <= data(nB);
                        nB :=7;
                        nP1 := 100;
                  -- Byte
                  elsif S_s=Byte then
                        nB := nB-1;
                        nBoutput <= data(nB);
                        elsif S_s=Pause1 then
                        output <= '0';
                        nP1 := nP1-1;
                        nB := 7;
                        nU :=9;
                        nS := 7;
                  end if;
end process;
```

```
end architecture;
```

Appendix C: Assembly [PIC] Code

```
; PIC.asm For 8x8Pixel Code
; Abraham Hsu
; April 2005
; MIT, Student

            list  p=18f452
            #include    "p18f452.inc"

;           __CONFIG    _CP_OFF&_WDT_OFF&_PWRTE_OFF&_RC_OSC

            radix hex
; Memory Map-------------------------------------------------


inCol equ         PORTB
inRow equ         PORTC
outCol      equ         PORTD

;inCol      equ         0xA0
;inRow      equ         0xA1
;outCol     equ         0xA2
;outRow     equ         0xA3
col         equ         0xA4
row         equ         0xA5
rowNum      equ         0xA6
reg         equ         0xA7
address     equ         0xA8
state equ         0xA9

row0  equ         0x80
row1  equ         0x81
row2  equ         0x82
row3  equ         0x83
row4  equ         0x84
row5  equ         0x85
row6  equ         0x86
row7  equ         0x87

stat0 equ         0x90
stat1 equ         0x91
stat2 equ         0x92
stat3 equ         0x93
stat4 equ         0x94
stat5 equ         0x95
stat6 equ         0x96
stat7 equ         0x97

id          equ         0xB0
nStart      equ         0xB1
nPause      equ         0xB2
x           equ         0xB3
```

```
;-------------------------------------------------------
            org         0x000
            goto  start

            org         0x004
            goto  start

; Program Starts
start
;Clear Registers-----------------------------------
            clrf  row0
            clrf  row1
            clrf  row2
            clrf  row3
            clrf  row4
            clrf  row5
            clrf  row6
            clrf  row7

            clrf  stat0
            clrf  stat1
            clrf  stat2
            clrf  stat3
            clrf  stat4
            clrf  stat5
            clrf  stat6
            clrf  stat7
;-----------------------------------------
            movlw 0x01
            movwf id

            ;inCol
            clrf  PORTB
            movlw 0xFF
            movwf TRISB
            clrf  PORTB ;portb input

            ;inRow
            clrf  PORTC
            movlw 0x00
            movwf TRISC
            clrf  PORTC ;portc output

            ;outCol
            clrf  PORTD
            movlw 0x00
            movwf TRISD
            clrf  PORTD ;portd output

;A0   CLK          In
;A1   History      Out
;A2   FPGA_in      In
;A3   Mode  In
;A4   TLGo  In
;A5   RightGo      Out
```

```
;E0   RightHis In
;E1   BottGo      Out
;E2   BottIn      In

            clrf  PORTA
            movlw b'00011101'
            movwf TRISA
            clrf  PORTA ;portd output

            clrf  PORTE
            movlw b'00000101'
            movwf TRISE
            clrf  PORTE ;portd outpu

;Testing Code----------------------------------------
;           goto  x7
;           bcf
;           movlw 0x01
;           movwf mode

;Mode SubRoutine-------------------------------------
mode  btfsc PORTA, 3    ;Mode=0?
            goto  x7          ;FALSE
            goto  input ;TRUE

;Input Program---------------------------------------
input movlw 0x01
            movwf col
            movwf row
            clrf  rowNum

            movf  row,w
            movwf inRow
            movf  stat0,w
            movwf state

input2      movf  col,w
            andwf state,w           ;value in WREG
            tstfsz      WREG  ;if (state0)
            goto  state1      ;FALSE
            goto  state0      ;TRUE

seeCol      movlw 0x80  ;if (col='7')
            cpfseq      col
            goto  addcolI     ;FALSE
                                    ;TRUE
            cpfseq      row         ;if (row='7')
            goto  addRowI     ;FALSE
            goto  output      ;TRUE
;states Subroutine-----------------------------------------------
state0      movf  col,w ;if (row1[col] = 0)
            andwf inCol,w
            xorwf col,w
            tstfsz      WREG  ;if (press=1?)
            goto  seeCol      ;FALSE
                                    ;TRUE
            call  compBit
```

```
            call  compStat
            goto  seeCol

state1      movf  col,w ;if (row1[col] = 0)
            andwf inCol,w
            tstfsz      WREG  ;if (press=0?)
            goto  seeCol      ;FALSE
                              ;TRUE
            call  compStat
            goto  seeCol
;compStat Subroutine-------------------------------------------
compStat movlw   0x90
            addwf   rowNum,w          ; unknown address is in WREG, how
to retrieve the data from this address?
            movwf   FSR1L, ACCESS     ; load address into FSR0L
            clrf    FSR1H, ACCESS     ; clear the four most significant
bits of FSR0

            call  getStat
            movf  col,w
            xorwf state,w           ;value in WREG
            movwf   INDF1, ACCESS  ; fetch value pointed to by FSR0

            return
;compBit Subroutine-------------------------------------------
compBit movlw   0x80
            addwf   rowNum,w          ; unknown address is in WREG, how
to retrieve the data from this address?
            movwf   FSR1L, ACCESS     ; load address into FSR0L
            clrf    FSR1H, ACCESS     ; clear the four most significant
bits of FSR0

            call  loadRow
            movf  col,w
            xorwf reg,w       ;value in WREG
            movwf   INDF1, ACCESS  ; fetch value pointed to by FSR0

            return
;getStat Subroutine-------------------------------------------
getStat     movlw   0x90
            addwf   rowNum,w          ; unknown address is in WREG, how
to retrieve the data from this address?
            movwf   FSR0L, ACCESS     ; load address into FSR0L
            clrf    FSR0H, ACCESS     ; clear the four most significant
bits of FSR0
            movf    INDF0, w, ACCESS  ; fetch value pointed to by FSR0

            movwf state
            return
;getRow Subroutine-------------------------------------------
loadRow     movlw   0x80
            addwf   rowNum,w          ; unknown address is in WREG, how
to retrieve the data from this address?
            movwf   FSR0L, ACCESS     ; load address into FSR0L
            clrf    FSR0H, ACCESS     ; clear the four most significant
bits of FSR0
```

```
            movf    INDF0, w, ACCESS  ; fetch value pointed to by FSR0

            movwf reg
            return

;Add Subroutines---------------------- --------------------------
addcolI     rlncf col
            goto  input2

addRowI     rlncf row
            incf  rowNum
            movf  row,w
            movwf inRow

            call  getStat

            movlw 0x01
            movwf col
            goto  input2
;Output Program---------------------------------------
output      movlw 0x01
            movwf col
            movwf row
            clrf  rowNum

            movf  row0,w
            movwf reg

output2     movf  col,w ;if (row1[col] = 0)
            andwf reg,w
            tstfsz      WREG
            call  light ;FALSE
                                ;TRUE
            movlw 0x80  ;if (col='7')
            cpfseq      col
            goto  addcol      ;FALSE
                                ;TRUE
            cpfseq      row         ;if (row='7')
            goto  addRow      ;FALSE
            goto  mode  ;TRUE
;Add Subroutines---------------------- --------------------------
addcol      rlncf col
            goto  output2

addRow      rlncf row
            incf  rowNum

            movlw 0x01
            movwf col
            call  loadRow
            goto  output2
;Light Subroutine-------------------------------------------
light movf  col,w
            movwf outCol
            movf  row,w
;           movwf outRow
            nop
```

```
            nop
;           call  delay
            clrf  outCol
;           clrf  outRow
            return
;Delay Subroutine-------------------------------------------------
;delay      clrf  TMR0
;again      btfss TMR0,3      ;This makes a delay of internal clk*16*2
;           goto  again
;           return
;Clock Subroutine-------------------------------------------------
;rising edge: 1)Make sure is in low 2)Wait for clk to be high again
clock btfsc PORTA, 0 ;clk=0?
            goto  clock ;FALSE
                                ;TRUE
clock2      btfss PORTA, 0 ;clk=1?
            goto  clock2      ;FALSE
            return                ;TRUE

;FPGA Output Subroutine-----------------------------------------
Foutput     movlw       b'01100100'
            movwf nPause
            movlw 0x08
            movwf nStart

            clrf  stat0
            clrf  stat1
            clrf  stat2
            clrf  stat3
            clrf  stat4
            clrf  stat5
            clrf  stat6
            clrf  stat7

Pause ;call clock
            btfsc PORTA, 2    ;FPGA_in=0?
            goto  Pause ;FALSE
                                ;TRUE
            decfsz      nPause      ;(nPause=0)?
            goto  Pause ;FALSE
                                ;TRUE
Start ;call clock
            btfss PORTA, 2    ;FPGA_in=1?
            goto  Foutput     ;FALSE
                                ;TRUE
            decfsz      nStart      ;(nPause=0)?
            goto  Start ;FALSE
                                ;True
;idTest SubRoutine-----------------------------------
            call x7

            movf  x           ;id=x?
            xorwf id,w
            tstfsz      WREG
            goto  Foutput     ;FALSE
            goto  write64     ;TRUE
;write64---------------------------------------------------------
```

```
write64     call  x7
            movf  x
            movwf row0

            call  x7
            movf  x
            movwf row1

            call  x7
            movf  x
            movwf row2

            call  x7
            movf  x
            movwf row3

            call  x7
            movf  x
            movwf row4

            call  x7
            movf  x
            movwf row5

            call  x7
            movf  x
            movwf row6

            call  x7
            movf  x
            movwf row7

            goto  output

;x SubRoutine-------------------------------------------------------
;Stores the next 8 bits in address x
x7          call  clock
            btfss PORTA, 2    ;FPGA_in=1?
            goto  x7F         ;FALSE
            bsf         x,7   ;TRUE
            goto  x6
x7F         bcf         x,7

x6          call  clock
            btfss PORTA, 2    ;FPGA_in=1?
            goto  x6F         ;FALSE
            bsf         x,6   ;TRUE
            goto  x5
x6F         bcf         x,6

x5          call  clock
            btfss PORTA, 2    ;FPGA_in=1?
            goto  x5F         ;FALSE
            bsf         x,5   ;TRUE
            goto  x4
x5F         bcf         x,5
```

```
x4          call  clock
            btfss PORTA, 2    ;FPGA_in=1?
            goto  x4F         ;FALSE
            bsf         x,4   ;TRUE
            goto  x3
x4F         bcf         x,4

x3          call  clock
            btfss PORTA, 2    ;FPGA_in=1?
            goto  x3F         ;FALSE
            bsf         x,3   ;TRUE
            goto  x2
x3F         bcf         x,3

x2          call  clock
            btfss PORTA, 2    ;FPGA_in=1?
            goto  x2F         ;FALSE
            bsf         x,2   ;TRUE
            goto  x1
x2F         bcf         x,2

x1          call  clock
            btfss PORTA, 2    ;FPGA_in=1?
            goto  x1F         ;FALSE
            bsf         x,1   ;TRUE
            goto  x0
x1F         bcf         x,1

x0          call  clock
            btfss PORTA, 2    ;FPGA_in=1?
            goto  x0F         ;FALSE
            bsf         x,0   ;TRUE
            return
x0F         bcf         x,0

            return

            end
```