Steps to Solve the Train Problem

1. Study the example state machine diagram(s) so that you understand exactly how it controls the trains with the states, inputs, and outputs.

2. If working on a PC at home, install the Altera software and obtain via email the secret code. Install the train project directory on your machine.

3. A zip file with the train project files is available for download on the web at:

   `www.ece.gatech.edu/users/hamblen/ALTERA/train.zip`

   Unzip to drive:/max2work/train. Where drive is the disk drive on which you installed the Altera CAD tools. If you don't have a zip file utility, get winzip free at `www.win95.com`

4. Read the example VHDL program and see how it implements the example state machine diagram.

5. Run the demo state machine simulation of the example state machine.

6. Read the problem statement and draw your own state diagram to solve the problem. Make up a name for each state.

7. Translate your state diagram into VHDL code by editing the example VHDL program, tcontrol.vhd. Only changes in the architecture section should be required since the inputs and outputs remain the same. You will only need to modify the type statement to reflect your new state names and will need a case in the state case statement for each of your states. The next state values in each case of the state case statement will test sensor inputs with a case or an if statement like the example program. Change the WITH..SELECT statements at the end of the program to generate the required outputs for each your states. In the editor the right mouse button can be used to pull up a VHDL template for most common statements. The main help menu also has many VHDL syntax examples. Compile the program and fix any syntax errors. Clicking on the error will send you to the error location in the editor.

8. Edit Tcontrol.vec, to add test cases for your new state machine. This will only require changes to the sensor input patterns. The number next to each pattern is the time at which it will change in the simulation. You can add more times and patterns, when needed.

9. Run the simulation and verify that your state machine is working properly. See running the simulation section for details.

10. Print out your simulation timing diagram – It is required as a prelab.

11. Compile train.vhd with your new tcontrol.vhd in the project directory. This adds the video output and simulation code to your state machine. See running the video train section for details.

12. Bring the train.sof and tcontrol.vhd files to the lab setup so that you can download it on the board and demo the video train operation to the lab TA.

13. Apply for Altera CPLD Certification, if your trains don’t crash.

VHDL State Machine Synthesis and Simulation using Altera CAD Tools and the UP1 CLPD Board

Introduction

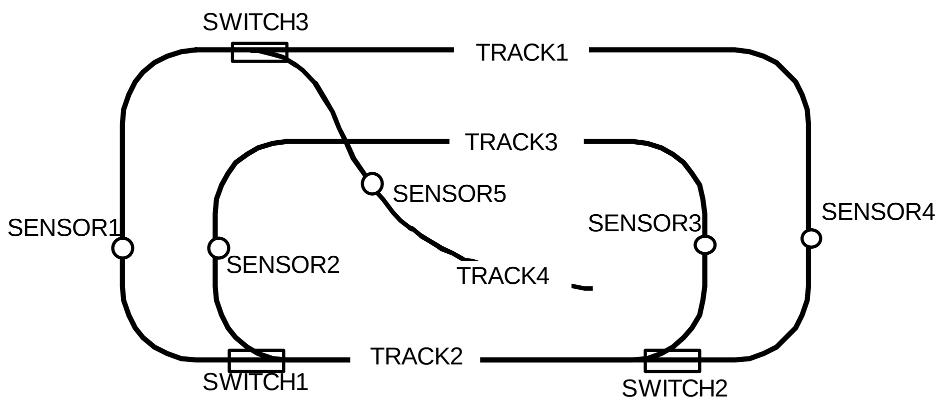
The purpose of this lab is to introduce the student to Complex Programmable Logic Devices, CPLD's, Logic Synthesis using VHDL and modern Computer Aided Design, CAD, tools for logic synthesis and simulation. VHDL is a hardware modeling language that has syntax similar to ADA or PASCAL. The entire process will be defined and illustrated in a step-by-step fashion in order to familiarize the student with the design process in the shortest possible time.

To facilitate this, a state machine design problem will be given along with a sample method for solving the problem. Examples will illustrate the methods used to convert the system design into a behaviorial VHDL model for the CPLD's CAD software. The VHDL synthesis tools automatically convert a VHDL description into a hardware design on a CPLD. The design can be simulated with actual time delays or downloaded into a CPLD. This CAD tool runs logic and state minimization so there is no need for K Maps or espresso. Altera has donated a CDROM student version of this tool. Which you can run on your home PC or you can use the professional versions installed in the lab. If you use the student version at home, your PC should have 16-64M memory and around 100M of disk space open. After installing the student version, it generates a unique code and you must request a matching secret code via email to enable the software. If you reinstall or change PCs, another code is required. This authorization process could take a day or two to get the email back, so start early if you want to run the student version at home. The professional version of this tool costs $5,000 per copy.

Design Setup

A state machine will be designed and programmed into an CPLD to operate the train system in a fashion that safely moves two trains without loss of life (or grade). This lab is based on a real HO gauge model train system that was used in the lab several years ago. We lost too many trains in wrecks, so it is now a virtual train system with a VGA video based display. Your VHDL state machine model, *tcontrol.vhd*, is combined with a larger VHDL model, *train.vhd*, which simulates and generates the VGA video display of the train system. The CPLD chip itself generates the VGA signal. A PC is used only to download the chip. The CPLD board has an Altera FLEX 10K20 CPLD chip that can be used for up to 20,000 gate designs.

Figure 1. Track Layout with Input Sensors and Output Switches and Output Tracks



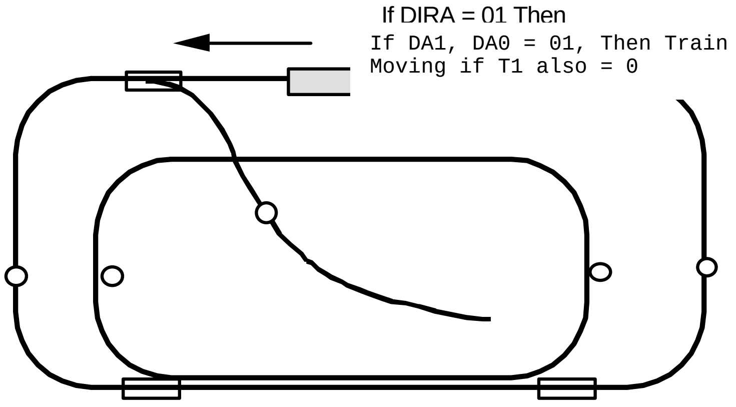
Track Power

There are two power sources assigned to the tracks (Power A, and Power B). Each track is either assigned power from source A (=0) or from source B (=1). In other words, if all four signals (T1, T2, T3, and T4) were asserted high, all tracks will be powered from source Y and would all be assigned the same Direction (see the next section under Track Direction for controlling train direction). See Figure 2 for an example. Note don't confuse power A and B with train A and B they are independent.

Track Direction

The direction for each track is controlled by four output signals (two for each power source), DA1 & DA0 for source A, and DB1 & DB0 for source B. When the direction signals indicate forward (01) on a particular power source then any train on a track assigned to that power source will move counterclockwise (on track 4, the train moves toward the outer track). When the signals imply reverse (10), the train(s) will move clockwise. The (11) value is illegal and should not be used. A "direction" indication set to stop (00) will stop any train assigned to the given source. (See Figure 2)

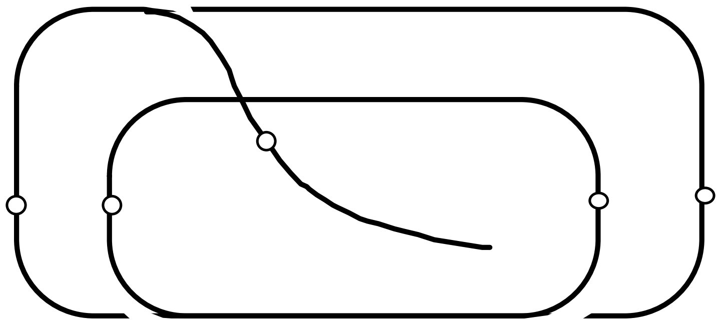
Figure 2. Track Power is Assigned from Two Sources: X and Y with Direction Independence.



Switch Direction

Switch directions are controlled by asserting SW1, SW2, and SW3 either high (outside connected with inside track) or low (outside tracks connected). That is, anytime all of the switches are set to 0, the tracks are setup such that the outside track forms a continuous circle. (See Figure 3). Tracks 3 and 4 cross at an intersection and care must be taken to avoid a crash where they cross. Just like a real train, if a train moves the wrong direction through an open switch it will derail. A switch has one track on one side and selects one of two tracks on the other side. If you go through the switch on the two track side, on the track is not connected to the one track side, it will derail.

Figure 3. Track Direction if all Switches Asserted (SW1 = SW2 = SW3 = 1)

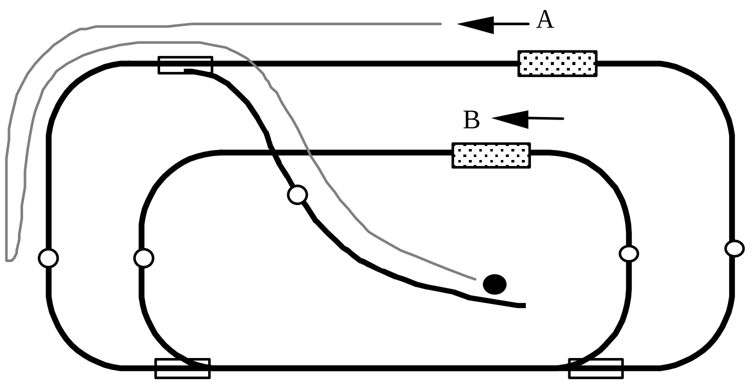


Problem Assignment

Your task is to design a state machine to operate the two trains avoiding collisions but minimizing their idle time. Trains must not crash by moving the wrong direction into an open switch. You must develop a simulation to verify your state machine is operating correctly before running the video train system for the final grade.

The trains are assumed to be in the initial positions as shown in Figure 4. Train A is to move counterclockwise around the outside track until it comes to Sensor 1, then move to the inside track stopping at Sensor 5 and waiting for B to pass Sensor 3 twice. Trains can move at different speeds. A train hitting a sensor can be stopped before entering the switch area.

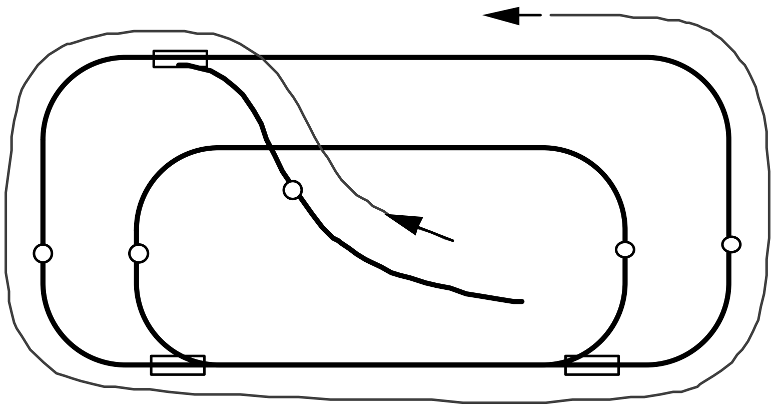
Figure 4. Initial Position of Trains at State Machine Reset with Initial Paths Designated



Once B has passed Sensor 3 twice, Train A moves to the outside track and continues around counter-clockwise until it picks up where it left off at the starting position.

Train B is to move as designated only stopping at a sensor to avoid collisions with A. Train B will then continue as soon as there is no potential collision and continue as designated.

Figure 5. Return Path of Train A



Train A and B should run continuously, stopping only to avoid a potential collision, until stopped by the TA.

CPLD Signal Summary

The CPLD will be programmed to implement the state machine described in the problem statement. The inputs and outputs are restricted to the specified pinouts as designated in

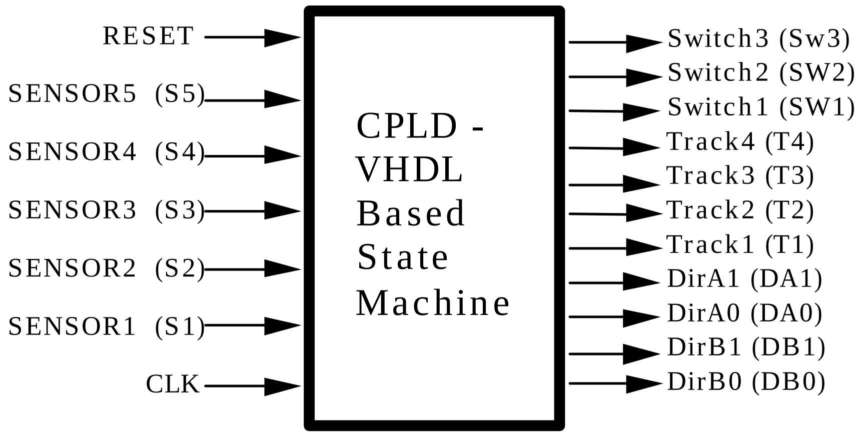
Figure 6. This insures signal compatibility with the lab system that controls the signals to and from the CPLD.

Input and Output Signals

It should be noted that sensors do not go high for only one cycle of the clock. In fact, sensors fire continuously for about ten clock cycles per passage of a train or car. This means that if your design is testing the same sensor from one state to another, you must wait for the signal to change from high to low.

The signal inputs and outputs have been summarized in the following figure:

Figure 6. Synopsis of CPLD Configuration



SWITCH (1,2,3)	= 0	Outside
	= 1	Inside
TRACK n (1,2,3,4)	= 0	A power on Track n
	= 1	B power on Track n
DIR (A,B)1 / DIR (A,B)0	= 00	Stop
	= 01	Foward
	= 10	Reverse
SENSOR (1,2,3,4,5)	= 0	Train not present
	= 1	Train present

Figure 7. Example Train ASM State Machine Design

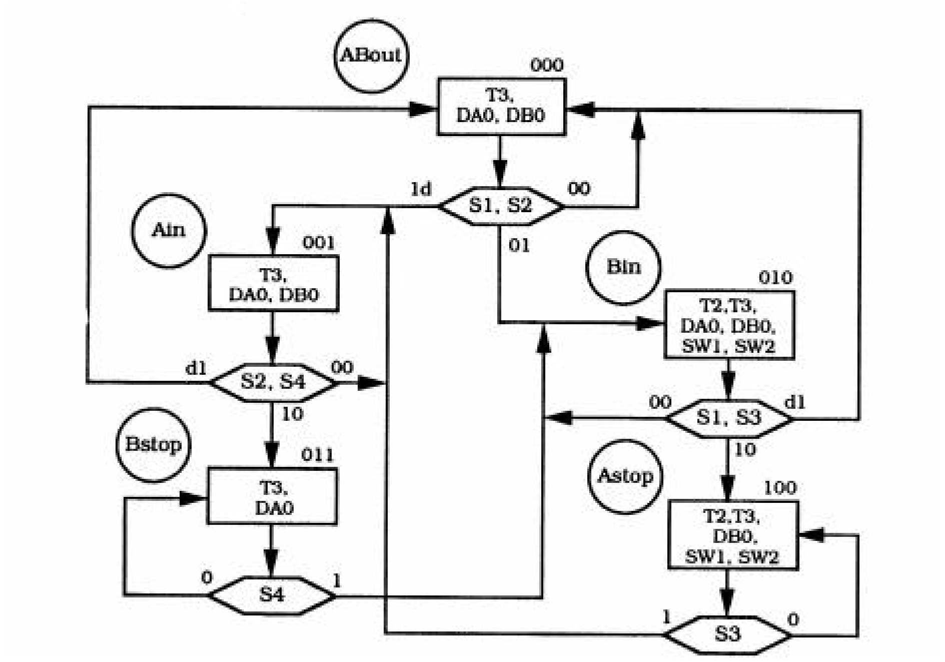
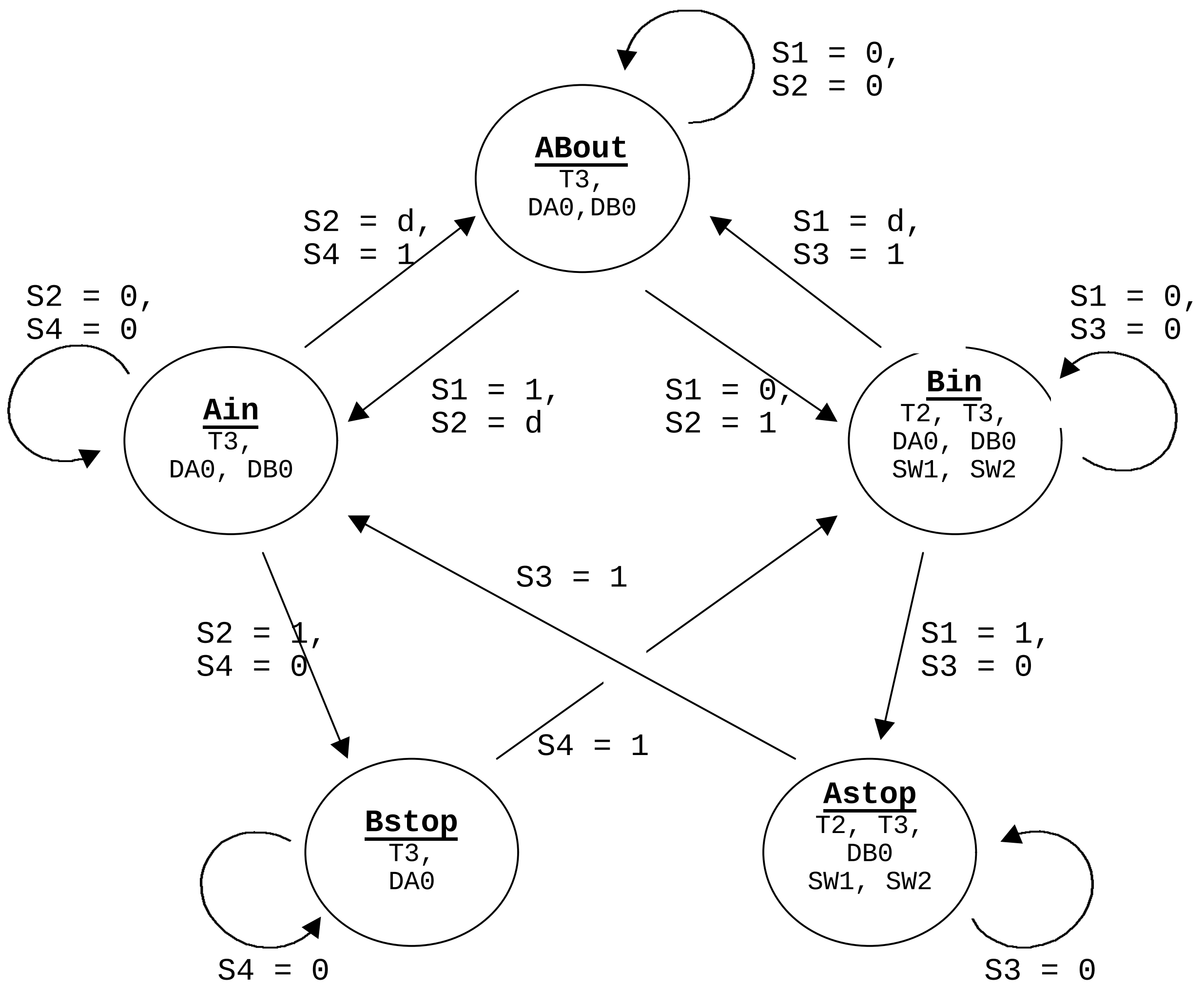


Figure 8. Example Train Bubble State Machine Design



This example state machine is somewhat simpler than the one in the problem statement. It runs the trains counterclockwise and avoids collisions with one train on the inner track and one train on the outer track. The ASM chart, another form of a state diagram, is shown in figure 7 for this problem. The state names, About, Ain, Bin. Bstop, Astop are in the circles. The rectangles contain the active outputs for that state. The input tests are in the diamond shapes. When two signals are shown in a diamond, they are both tested at the same time for the indicated values. A state diagram is shown in figure 8. Figure 7 and figure 8 contain the same information, they are just different styles of state diagrams.

Description of States in Example State Machine

All States

- T3 Asserted, the B Power Supply is assigned to track 3,

About: "Trains A and B Outside"

- DA0 Asserted: Train A is on the outside track and moving counter-clockwise (forward),
- DB0 Asserted: Train B is on the inner track (not the common track) and also moving forward,
- Note that by NOT Asserting DA1, it is automatically zero -- same for DB1. Hence, the pair DA1, DA0 = 0, 1. The pair DB1, DB0 = 0, 1.

Ain: "Train A moves to Common Track"

- Sensor 1 has fired either first or at the same time as Sensor 2.
- Either Train A is trying to move towards the common track or
- Both trains are attempting to move towards the common track.
- Both trains are allowed to enter here; however, state Bstop will stop B if both have entered.
- DA0 Asserted: Train A is on the outside track and moving counter-clockwise (forward),
- DB0 Asserted: Train B is on the inner track (not the common track) and also moving forward,

Bstop: "Train B stopped at S2 waiting for Train A to clear common track"

- DA0 Asserted: Train A is moving from the outside track to the common track.
- Train B has arrived at Sensor 2 and is stopped and waits until Sensor 4 fires.
- SW1 and SW2 are NOT Asserted to allow the outside track to connect to common track.
- Note that T2 is not asserted making Track 2 tied to the A Power Supply.

Bin: "Train B has reached Sensor 2 before Train A reaches Sensor 1"

- Train B is allowed to enter the common track. Train A is approaching Sensor 1.
- DA0 Asserted: Train A is on the outside track and moving counter-clockwise (forward),
- DB0 Asserted: Train B is on the inner track moving towards the common track,
- SW1 Asserted: Switch 1 is set to let the inner track connect to the common track.
- SW2 Asserted: Switch 2 is set to let the inner track connect to the common track.
- T2 Asserted: the B Power Supply is also assigned to the common track.

Astop: "Train A stopped at S1 waiting for Train B to clear the common track"

- DB0 Asserted: Train B is on the inner track moving towards the common track,

- SW1 and SW2 Asserted: Switches 1 and 2 are set to let the inner track connect to the common track.
- T2 Asserted: the B Power Supply is also assigned to the common track.

Note: All signals that are not "Asserted" are zero and imply a logical result as described.

The corresponding VHDL code for this state machine is shown below. You will need to modify and expand this code for your state machine problem solution. Inputs sensors and output control signals will be the same but the states will change. For additional VHDL help, see the help files in the Altera CAD tool or look at the VHDL tutorial, which can be found on the GT Altera web page.

```
--
--
-- Example State machine to control trains
-- File: Tcontrol.vhd
--
-- These libraries are required in all VHDL source files
library IEEE;
use  IEEE.STD_LOGIC_1164.all;
use  IEEE.STD_LOGIC_ARITH.all;
use  IEEE.STD_LOGIC_UNSIGNED.all;

-- This section defines state machine inputs and outputs
-- No modifications should be needed in this section
ENTITY Tcontrol IS

      PORT(reset, clock, sensor1, sensor2, sensor3, sensor4, sensor5: in
std_logic;
     switch1, switch2, switch3: out std_logic;
     track1, track2, track3, track4: out std_logic;
-- dirA and dirB are 2-bit logic vectors(i.e. an array of 2-bits)
     dirA, dirB : out std_logic_vector(1 Downto 0));
END Tcontrol;

-- This code describes how the state machine operates
-- This section will need changes for your state machine
ARCHITECTURE a OF Tcontrol IS
-- Define local signals (i.e. non input or output signals) here
      TYPE STATE_TYPE IS (ABout,Ain,Bin,Astop,Bstop);
      SIGNAL state: STATE_TYPE;
    SIGNAL sensor12, sensor13, sensor24: std_logic_vector(1 Downto 0);

BEGIN
-- This section describes how the state machine behaves

-- Run this process once every time the clock changes
      PROCESS (clock)
      BEGIN
-- Reset to this state (i.e. asynchronous reset)
            IF reset = '1' THEN
                  state <= ABout;
            ELSIF clock'EVENT AND clock = '1' THEN
-- clock’EVENT means value of clock just changed
--This section will execute once on each positive clock edge
--Signal assignments in this section will generate D flip flops
--
-- Case statement to determine next state
                  CASE state IS
                        WHEN ABout =>
-- This Case statement checks both sensor1 and sensor2 bits
                              CASE Sensor12 IS
-- Note: VHDL’s use of double quote for bit vector versus
-- a single quote for only one bit!
```

```
                              WHEN "00" => state <= About;
                              WHEN "01" => state <= Bin;
                              WHEN "10" => state <= Ain;
                              WHEN "11" => state <= Ain;
-- Default case is always required
                              WHEN OTHERS => state <= ABout;
                              END CASE;

                        WHEN Ain =>
                              CASE Sensor24 IS
                              WHEN "00" => state <= Ain;
                              WHEN "01" => state <= ABout;
                              WHEN "10" => state <= Bstop;
                              WHEN "11" => state <= ABout;
                              WHEN OTHERS => state <= ABout;
                              END CASE;

                        WHEN Bin =>
                              CASE Sensor13 IS
                              WHEN "00" => state <= Bin;
                              WHEN "01" => state <= ABout;
                              WHEN "10" => state <= Astop;
                              WHEN "11" => state <= About;
                              WHEN OTHERS => state <= ABout;
                              END CASE;

                        WHEN Astop =>
                              IF Sensor3 = '1' THEN
                                    state <= Ain;
                              ELSE
                                   state <= Astop;
                              END IF;

                        WHEN Bstop =>
                              IF Sensor4 = '1' THEN
                                    state <= Bin;
                              ELSE
                                    state <= Bstop;
                              END IF;

                  END CASE;
            END IF;
      END PROCESS;

-- combine sensor bits for case statements above
-- “&” operator combines bits
sensor12 <= sensor1 & sensor2;
sensor13 <= sensor1 & sensor3;
sensor24 <= sensor2 & sensor4;


-- These outputs do not depend on the state for this state machine
    Track1  <= '0';
    Track4  <= '0';
    Switch3 <= '0';

-- Outputs that depend on state, use state to select value

-- Be sure to specify every output for every state
-- values will not default to zero!
      WITH state SELECT
```

```
            Track3      <=    '1'   WHEN ABout,
                        '1'   WHEN Ain,
                        '1'   WHEN Bin,
                        '1'   WHEN Astop,
                        '1'   WHEN Bstop;
      WITH state SELECT
            Track2      <=    '0'   WHEN ABout,
                        '0'   WHEN Ain,
                        '1'   WHEN Bin,
                        '1'   WHEN Astop,
                        '0'   WHEN Bstop;
      WITH state SELECT
            Switch1 <=  '0'   WHEN ABout,
                        '0'   WHEN Ain,
                        '1'   WHEN Bin,
                        '1'   WHEN Astop,
                        '0'   WHEN Bstop;
      WITH state SELECT
            Switch2 <=  '0'   WHEN ABout,
                        '0'   WHEN Ain,
                        '1'   WHEN Bin,
                        '1'   WHEN Astop,
                        '0'   WHEN Bstop;
      WITH state SELECT
            DirA  <=    "01"  WHEN ABout,
                        "01"  WHEN Ain,
                        "01"  WHEN Bin,
                        "00"  WHEN Astop,
                        "01"  WHEN Bstop;
      WITH state SELECT
            DirB  <=    "01"  WHEN ABout,
                        "01"  WHEN Ain,
                        "01"  WHEN Bin,
                        "01"  WHEN Astop,
                        "00"  WHEN Bstop;
END a;
```

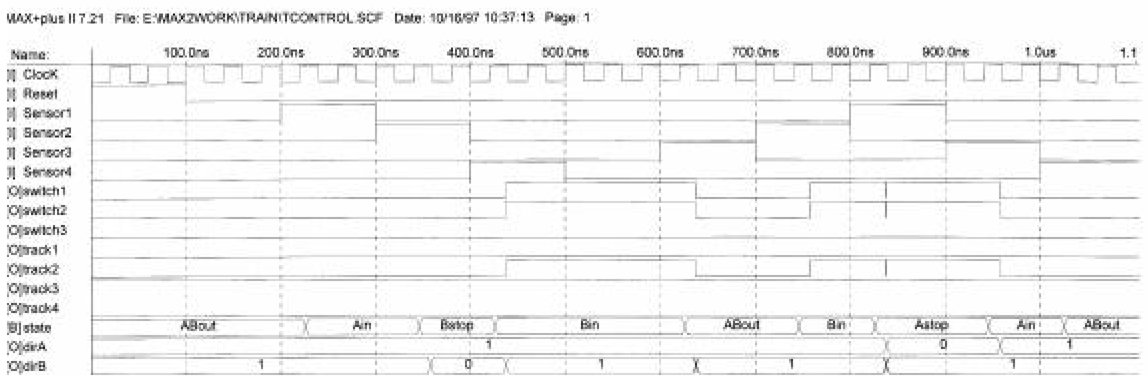
Simulation Vector file for State Machine Simulation

This file, *tcontrol.vec*, controls the simulation and tests the state machine. This file sets up a 40ns clock and specifies sensor patterns, inputs to the state machine, which will be used to test the state machine. You will need to change the sensor input patterns to test your new state machine. The numbers left of the ">" are the time at which the patterns change. As an example "INPUT Sensor1, PATTERN, 0>0" means that the Sensor1 input is a zero at time zero and 100>1 would change to a one at time 100ns in the simulation. These patterns were chosen by picking a path in the state diagram that moves to all of the different states. Sensor inputs should not change faster than the clock cycle time of 40ns. It would not be a bad idea to test all states and arcs in your state machine simulation.

```
% Simulation Test Vectors for Train Control State Machine - File: Tcontrol.vec %
START 0us;
STOP 1100ns;
```

```
% Generate Clock %
INTERVAL 20ns;
INPUTS ClocK;
PATTERN
0 1;
% Reset State Machine %
INPUTS Reset;
PATTERN
0>1
100>0;
% Cycle Sensor Inputs at following times to cycle through states%
INPUTS Sensor1;
PATTERN
0>0
100>0
200>1
300>0
400>0
500>0
600>0
700>0
800>1;
INPUTS Sensor2;
PATTERN
0>0
100>0
200>0
300>1
400>0
500>0
600>0
700>1;
INPUTS Sensor3;
PATTERN
0>0
100>0
200>0
300>0
400>0
500>0
600>1
700>0
800>0
900>1
1000>0;
INPUTS Sensor4;
PATTERN
0>0
100>0
200>0
300>0
400>1
500>0
600>0
700>0
800>0
900>0
1000>1;
% Display These Signals on Timing Diagram %
BURIED state; % Buried means it is an internal signal - not an input or output %
OUTPUTS switch1 switch2 switch3 track1 track2 track3 track4 dirA dirB;
```

Figure 9. Altera Simulation of Tcontrol.vhd using Tcontrol.vec vector file



Running the Train Control Simulation on the Altera CAD tools

Make Tcontrol.vhd the current project with *File -> Project -> Name*
Then find and select Tcontrol.vhd.

Select *File-> Project -> Save Compile and Simulate.*
The simulator will run automatically if there are no compile errors. Select Open SCF to see the timing diagram display of your simulation as seen in figure 9. Whenever you change your VHDL source you need to repeat this step. If you get compile errors, clicking on the error will move the text editor to the error location. The Altera software has extensive on-line help including VHDL syntax examples.

Make any text changes to Tcontrol.vhd or Tcontrol.vec (test vector file) with *File - > Open* . This brings up a special editor window – note that the menus at the top of the screen change depending on which window you are currently in.

To update the simulation with new test vectors from a modified Tcontrol.vec file Select *File -> Open* Tcontrol.scf or select the timing display window if it is already open. If you have changed Tcontrol.vec update the scf file with *File -> Import Vector File*. You can then hit Start on the simulation window to rerun the simulation with new test vectors.

Running the Video Train System (After Simulation Works)

Make Train.vhd the current project with *File -> Project -> Name*
Then find and select Train.vhd. Train.acf must be in the project directory since it contains the FLEX chip pin assignment information.

Select *File-> Project -> Save and Compile*. Train.vhd will link in your tcontrol.vhd file that is in the same directory, when it is compiled. This is a big program so it will take a few minutes to compile. Try running on a fast machine with 32M or more of memory.

Select *MaxPlus -> Programmer*. Pull JTAG menu at top to see that Multi-Device is turned on. In *JTAG-> multi device JTAG chain setup* select EPF10K20 as the device and train.sof as the programming file to download. In case of problems, see UP1 board manual pages 20-22 for more details. A copy is included on the following page. Under *options -> hardware* select byteblaster and lpt2. Note: Jumpers must be set for flex only on UP1 board, power supply must be connected, and cable must be plugged into printer port. When everything is setup, the configure button in the programming window should highlight. To download the board click on the highlighted configure button.

Train output should appear on the VGA monitor after downloading is complete. Flex PB1 is run/step and Flex PB2 is reset (note: these are the pushbuttons to the right and top of the board as seen in figure 10).

If a train wreck is detected the simulation halts and the monitor will flash. Sensor and switch values are indicated with a green or red square on the display. Green indicates no train present on a sensor and it indicates switch closed for a switch. The left FLEX seven segment LED shows the values of the track signals in hexadecimal and the right led indicates the values of DIRA and DIRB in hexadecimal. The octal Flex dip switch controls the speed of Train A(low 4-bits) and B(high 4-bits).

Detailed Information Needed only to Download and Run the Design on the CPLD on the Altera UP1 Board

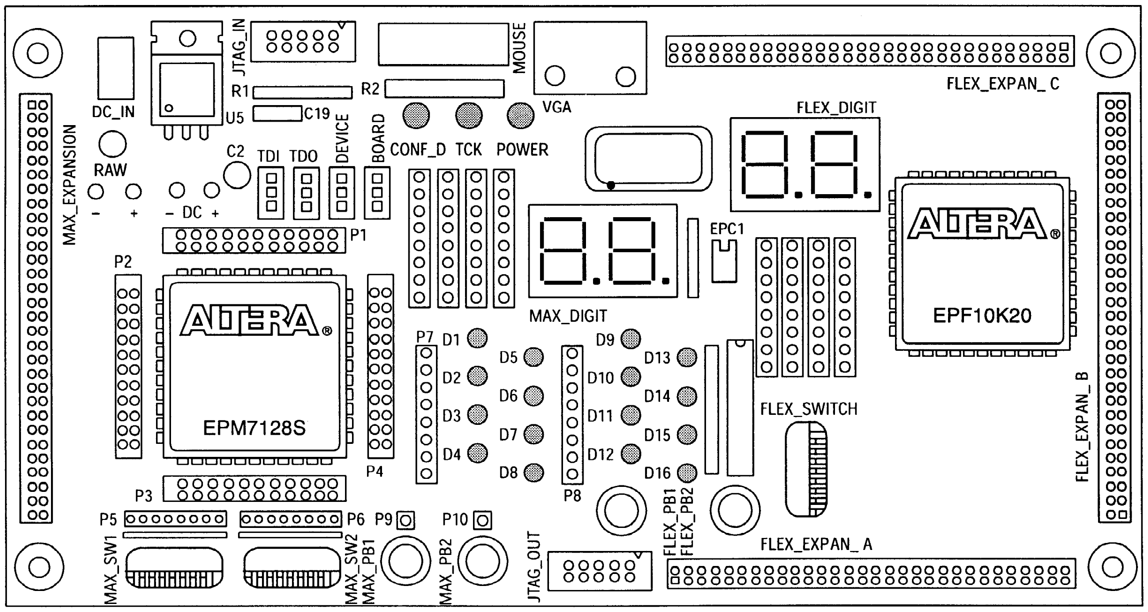


Figure 10. Altera UP1 CLPD Demo Board

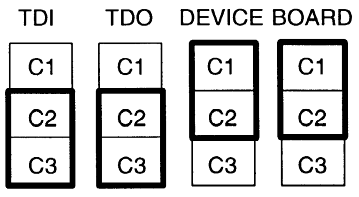
EPF10K20 Configuration

This section describes the procedures for configuring only the EPF10K20 device, (i.e., how to set the on-board jumpers, connect the ByteBlaster download cable, and set options in the MAX+PLUS II software).

Setting the On-Board Jumpers for EPF10K20 Configuration

To configure only the EPF10K20 device in a JTAG chain, set the jumpers TDI, TDO, DEVICE, and BOARD as shown in Figure 8.

Figure 8. Jumper Settings for Configuring Only the EPF10K20 Device



Connecting the ByteBlaster Download Cable for EPF10K20 Configuration

Attach the ByteBlaster directly to the PC's parallel port and to the JTAG_IN connector on the UP 1 Education Board. For more information on setting up the ByteBlaster, go to the *ByteBlaster Parallel Port Download Cable Data Sheet*.

Setting the JTAG Options in MAX+PLUS II for EPF10K20 Configuration

The following steps describe how to use MAX+PLUS II to configure the EPF10K20 device in a JTAG chain. For more information on how to configure a device, see MAX+PLUS II Help.

1. To configure more than one EPF10K20 device, turn on the *Multi-Device JTAG Chain* command (JTAG menu) in the MAX+PLUS II Programmer.

2. Choose **Multi-Device JTAG Chain Setup** (JTAG menu).

3. In the **Multi-Device JTAG Chain Setup** dialog box, select *EPF10K20* in the *Device Name* drop-down list box.

4. Type the name of the programming file for the EPF10K20 device in the *Programming File Name* box. The **Select Programming File** button can also be used to browse your computer 's directory structure to locate the appropriate programming file.

5. Choose **Add** to add the device and associated pr ogramming file to the *Device Names & Programming File Names* box. The number to the left of the device name shows the or der of the device in the JTAG chain. The device's associated pr ogramming file is displayed on the same line as the device name. If no pr ogramming file is associated with a device, "<none>" is displayed next to the device name.

6. Choose **Detect JTAG Chain Info** to have the ByteBlaster check the device count, JTAG ID code, and total instr uction length of the JTAG chain. A message just above the **Detect JTAG Chain Info** button reports the information detected by the ByteBlaster . You must manually verify that this message matches the information in the *Device Names & Programming File Names* box.

7. To save the current settings to a JCF for futur e use, choose **Save JCF**. In the **Save JCF** dialog box, type the name of the fi le in the *File Name* box and then select the desir ed directory in the *Directories* box. Choose **OK**.

8. Choose **OK** to save your changes.

9. In the MAX+PLUS II Programmer, choose **Configure**.