

1.1 Problem Statement

A manufacturer of frozen goods produces a frozen banana fruit treat. However, some of the bananas are damaged during the freezing process, and suffer from the condition of “freezer-burn”. Due to the large volume of bananas the manufacturer must process, a system is required to sort the bananas between good and bad. The goal of the system is to take an input of unsorted frozen bananas and automatically separate the damaged bananas from the good bananas.

1.2 Specifications

Functional

- Bananas are loaded into the system via a loading bay by a human operator.
- Bananas are moved one at a time from the loading bay to a position on the conveyer belt under an optical sensor which detects the color of the banana.
- Bananas are sorted on the basis of color. Good bananas are yellow, while bananas damaged by freezer burn are blue.
- The system performs calibration of the optical sensor input upon system initialization.
- The system performs automatic recalibration of the optical sensor input while the system is running using an adaptive algorithm that tracks and counters the effects of sensor drift.
- Good bananas are sent forward along the conveyer belt to a product collection bay.
- Damaged bananas are sent backward along a conveyer belt to a waste collection bay.
- The distance the conveyor belt travels is controlled using an angle sensor as a feedback mechanism.
- When the conveyor belt is moving backwards, an alarm is sounded to alert nearby human operators.
- Good bananas are collected from the product collection bay by a human operator.
- Bad bananas are collected from the waste collection bay by a human operator.
- The system counts and displays the number of damaged bananas processed since system initialization.
- A manual over-ride is provided to allow a human operator to interrupt and suspend conveyor belt operation.

Performance

- A maximum of 5% of good bananas will be incorrectly identified as damaged.
- A maximum of 1% of bad bananas will be incorrectly identified as good.
- The processing rate is 5 bananas per minute.

Environment

- The system performs I/O of analog signals to and from the conveyor belt via the customer-provided UNB Interface Board.
- The system interfaces with the customer-provided LEGO conveyor belt.

2.1 System Design

The system was developed to control the operation of the customer-provided conveyor belt. Control of the conveyor belt motor and buzzer is applied using input signals from an angle sensor and an optical sensor. The angle sensor provides feedback about motor movement from which the distance travelled by the conveyor can be determined. The optical sensor allows bananas to be detected and identified as good or bad as they are carried past the optical sensor by the conveyor. The system interfaces with the angle sensor, optical sensor, motor and buzzer of the conveyor belt through the UNB interface board. The interface board performs signal conditioning and power regulation.

2.1.1 System Block Diagram

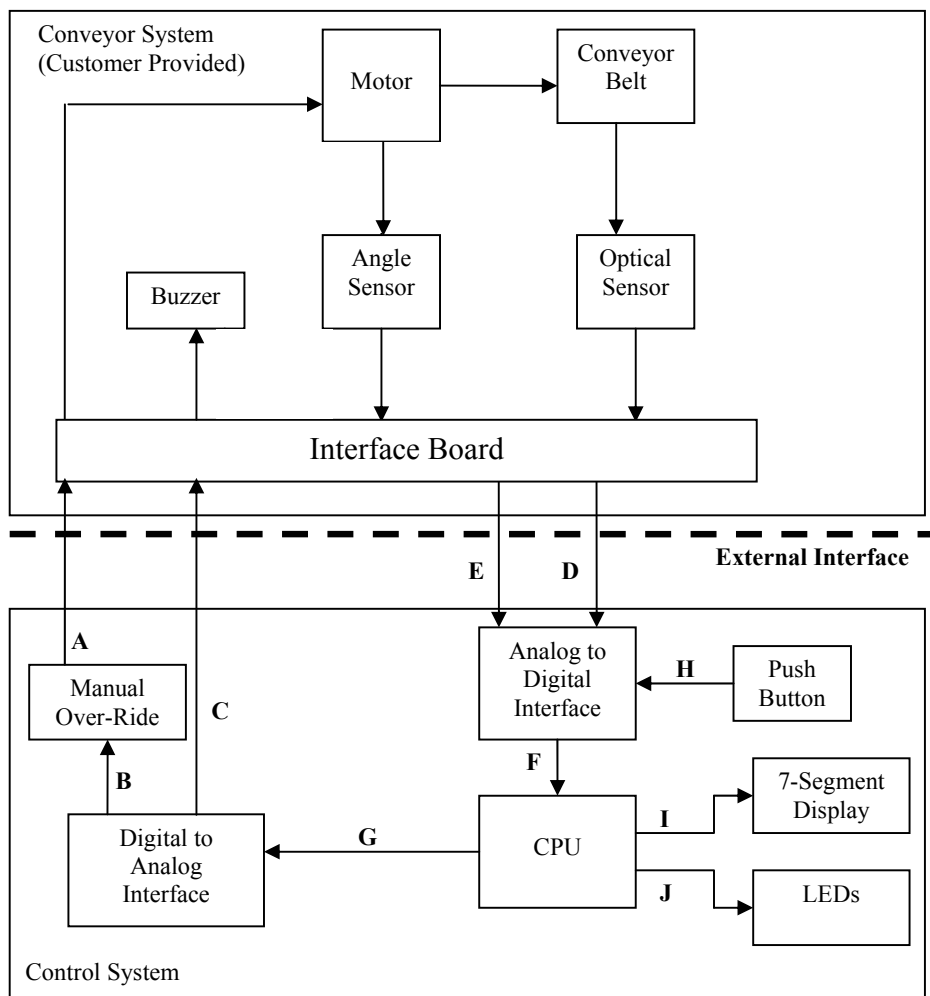


Figure 1: System Block Diagram

- A. A mechanical toggle switch, connected to Bipolar DAC output (+9V/-9V) when closed and open-circuit when opened.
- B. Voltage from the Bipolar DAC amplified to the range -9V to +9V.
- C. Voltage from the Unipolar DAC amplified to the range 0V or 9V.
- D. Optical sensor signal ranging from -0.7V to 5.1V (voltage clipping using a zener diode).
- E. Angle sensor signal consisting of four different quantized voltages, ranging from -0.7V to 5.1V (voltage clipping using a zener diode).
- F. 8-bit digital value of the sampled signal from the optical sensor, angle sensor, or pushbutton.
- G. 8-bit digital value representing the speed and direction of the motor or 8-bit value representing the volume of the buzzer.
- H. Pushbutton voltage value of either 0V or 5V (5V when button is pushed).
- I. 4-bit digital value representing a number from 0-9.
- J. 4-bit digital value representing the state of 4 LEDs. A bit-value of 0 turns on the LED.

2.1.2 Functional Description of Modules

The system modules shown in Figure 1 have the following functions:

Analog to Digital Interface:

- Converts the analog angle sensor signal to an 8-bit digital value
- Converts the analog optical sensor signal to an 8-bit digital value

Manual Over-ride:

- When switched on, power to the motor is cut off to stop the conveyor belt

Digital to Analog Interface:

- Converts the speed and direction of the motor from an 8-bit digital value to an analog voltage
- Converts the volume of the buzzer from an 8-bit digital value to an analog voltage

CPU:

- Encompasses the processor, RAM, EEPROM and supporting circuitry required to execute user code that controls the conveyor system
- Accepts DAC converted data of the angle and optical sensor signals
- Based on the input signals, the CPU controls the direction and speed of motor as well as the buzzer status

Push Button:

- Used to engage calibration of the optical sensor
- When pressed, the current value of the optical sensor is stored into memory

7-Segment Display:

- The four bits are used to determine the value displayed by the seven-segment display
- Displays the number of bad bananas discarded by the system

LEDs:

- The four bits are used to determine the state of four LEDs
- Serves as a progress bar during the calibration of the optical sensor

2.1.3 ASM Diagram

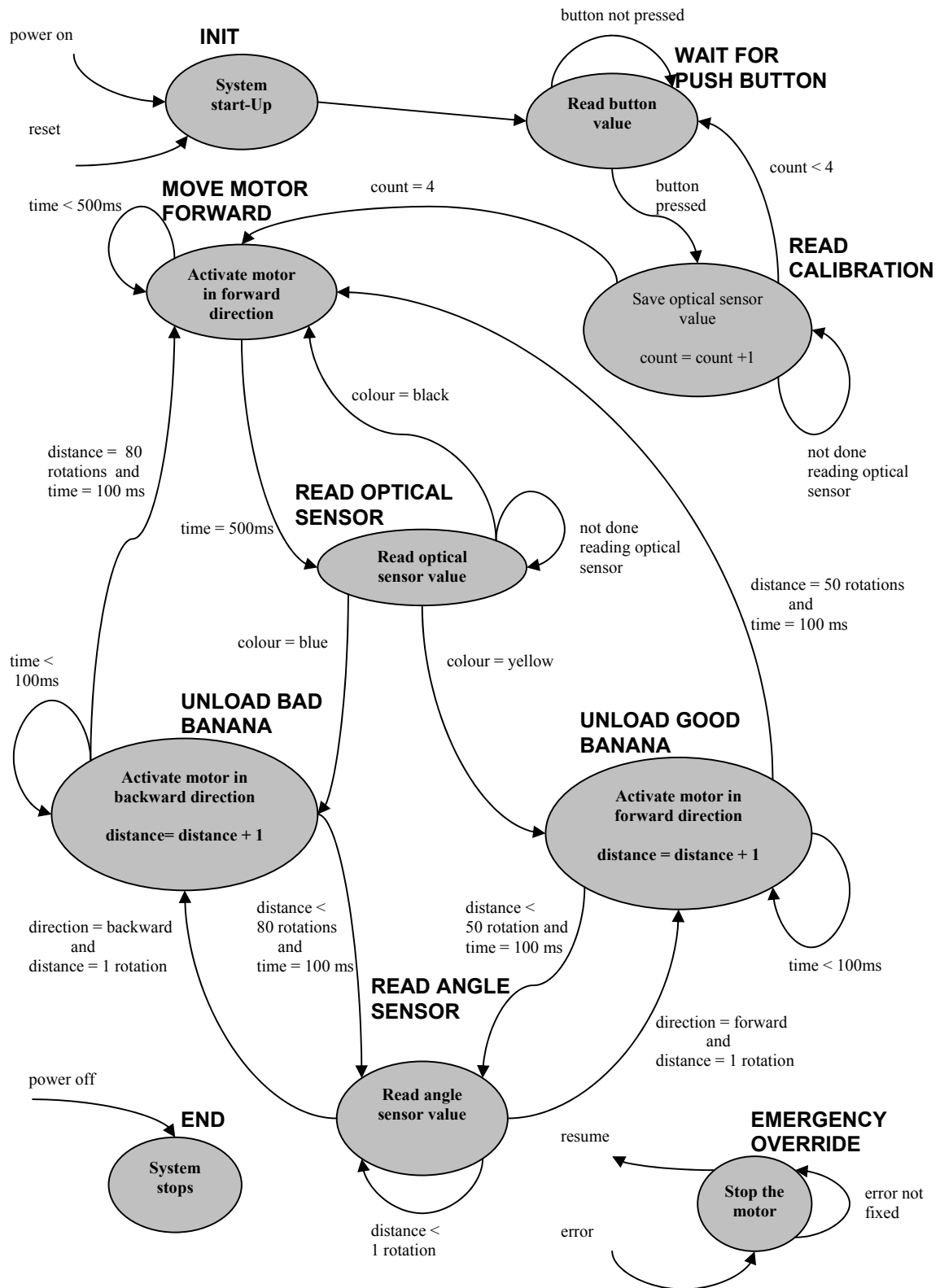


Figure 2: Algorithmic State Machine Diagram

ASM Diagram Notes:

- The emergency override state can be entered from any other state when the override switch is engaged. This disables the conveyor motor for the duration of the state.
- The system is meant to run continuously, but in the event that the power is turned off, the system will enter into the End state. This state can be entered from any other system state. In order to restart the system, the power must be turned back on and the system will enter the Init state.
- If the system is reset at any time, it will return to the Init state. This reset condition can be invoked from any state.

2.2 Hardware Design

The requirements for the system called for the development of an embedded microprocessor system for controlling the conveyor belt. A perforation board was used to hold the various digital IC chips and analog components of the system. The system buses and signal lines were created by wire-wrapping.

The starting point of the system was the UNB SBC188 daughter board which houses an Intel 80C188XL microprocessor and exposed the pin-outs of the CPU chip for wire-wrapping purposes. The 80C188XL CPU uses a 20-bit logical address bus and an 8-bit logical data bus. The physical bus for data and address is shared in the form of an 8-bit AD bus that carries the 8 least significant bits of the address bus as well as the data bus. Demultiplexing of the address and data buses is performed using an 8-bit latch.

Overview of Core System Components:

A 32 KB EEPROM (28H256) is used to store software code to be executed by the system. Hex code was generated from C code using the Paradigm C++ software tool and its accompanying Locate utility. The hex code was written onto the EEPROM using an EEPROM programmer.

The system uses 8KB of SRAM (HM6264) for memory. This small amount of RAM proved to be sufficient to execute the developed software.

Analog signals are inputted to the system using a 4-channel 8-bit ADC which converts voltages in the range of 0V to +5V to 8-bit digital values. The ADC is used to sample input from the angle sensor, the optical sensor, and the calibration push button.

Analog signals are outputted from the system using two 8-bit DACs. One DAC produces output voltage in the range of -9V to +9V with the use of a dual-stage op-amp circuit. This voltage is used to drive the DC motor of the conveyor belt. The direction in which the motor turns depends on the polarity of the applied voltage. The second DAC produces output voltage in the range of 0V to +9V using a single-stage op-amp circuit. This voltage is used to control the volume of the alarm buzzer.

2.2.1 Hardware Block Diagram

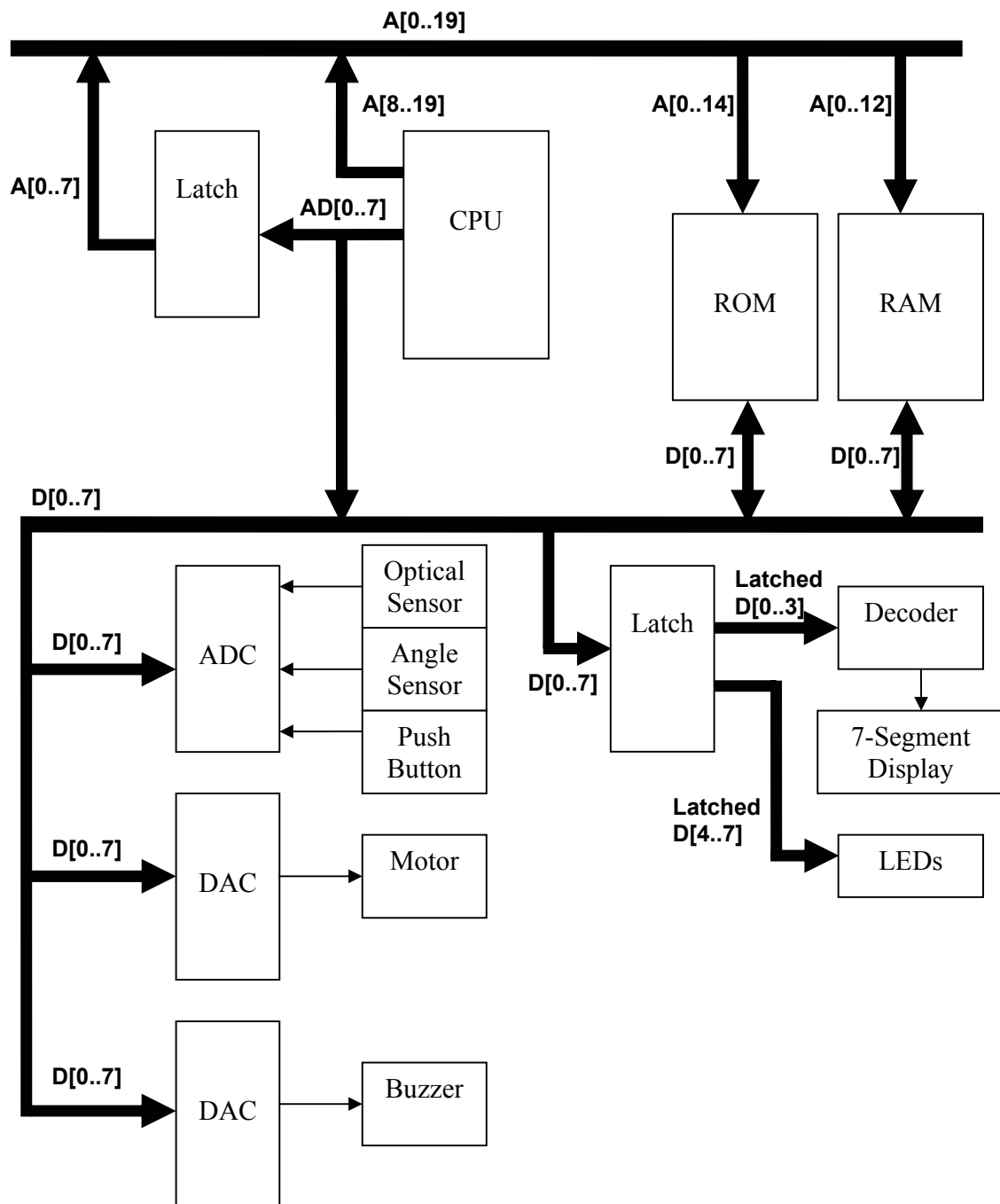


Figure 3: Hardware Block Diagram

2.2.4 Board Layout Diagram

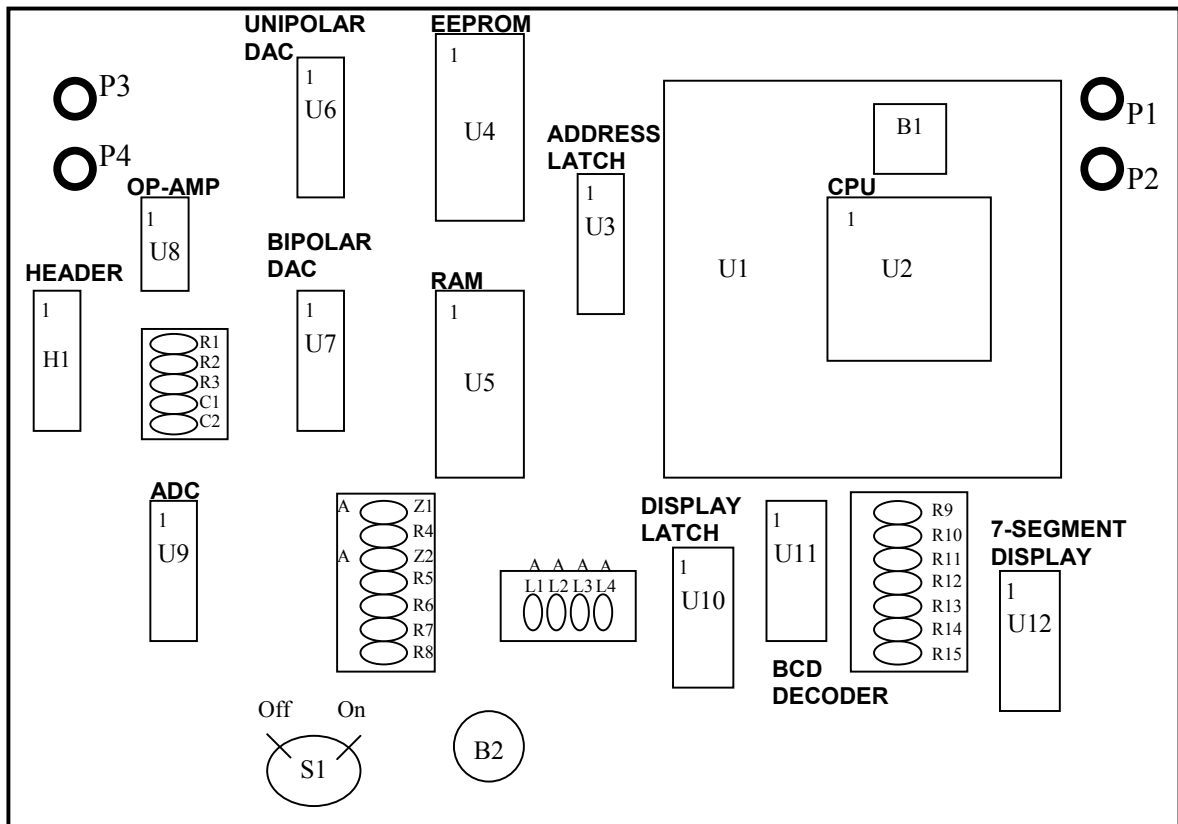


Figure 5: Board Layout

2.2.5 Parts List

Integrated Circuit Chips:

Note that a “1” on the board layout diagram indicates the position of the chip’s pin 1.

- U1 – UNB SBC188 daughter board
- U2 – Intel 80C188XL microprocessor
- U3 – Fairchild Semiconductor DM74LS373N Latch
- U4 – Xicor X28HC256 EEPROM (32Kb)
- U5 – Hitachi HM6264B RAM (8Kb)
- U6, U7 – National Semiconductor 0832LCN DAC
- U8 – Texas Instruments TL084CN OpAmp
- U9 – National Semiconductor 0844CCN ADC
- U10 – SAM KS74HCTL374N Latch
- U11 – Texas Instruments SN7447AN Decoder
- U12 – Lite-on LTS-312AR Seven-Segment Display

Buttons:

- B1, B2 – momentary push button

Switches:

- S1 – Toggle switch

Diodes:

Note that a “A” on the board layout diagram indicates the position of the diode’s anode.

- Z1, Z2 – 5.1V Zener Diode 1N751
- L1, L2, L3, L4 – Light Emitting Diode

Resistors:

- R1, R2 – 20K Ω
- R3 – 10K Ω
- R4, R5, R6, R7 – 330 Ω
- R8, R9, R10, R11, R12, R13, R14, R15 – 270 Ω

Capacitors:

- C1, C2 – 10 μ F

Power Connections

- P1 – +5V
- P2 – GND
- P3 – +12V
- P4 – -12V

Headers:

- H1 – 20 pin header

2.2.6 Hardware Design Decisions

7-Segment Display and LEDs:

In order to minimize the number of chips needed in the system, the seven-segment display and the four light emitting diodes share the same data latch. This was possible due to the fact that the seven-segment display uses a 4-to-7 BCD decoder to control the value displayed. For example, when the value 1001_b is entered into the decoder, the seven-segment display shows a “9”. Thus the decoder only uses 4-bits of the 8-bit latch, allowing the four LEDs use the remaining bits. In this system, the four least significant bits are used for the decoder while the four most significant bits control the LEDs.

Angle Sensor:

The angle sensor is used to determine the number of rotations the motor has rotated through. Knowing the number of rotations performed allows the system to determine the distance the conveyor has moved. An alternative approach to moving the conveyor a specified distance would have been to use a software timer. For example, activating the motor for 100 rotations is equivalent to activating the motor for 8 seconds. However, in the event that the system is interrupted by an emergency override, a software timer would fail to gauge the correct distance travelled. The use of the angle sensor allows the exact distance to be measured, regardless of the speed of the motor or if the motor is stopped by the user.

Emergency Override:

The emergency override was implemented using a mechanical toggle switch. When the switch is thrown into the off position, the voltage to the motor is immediately cut off without shutting the system down. Using a mechanical switch as opposed to interrupts results in a more reliable emergency override. Even in the event of a software failure, the emergency override will function. The use of the emergency override is better than powering off the system completely because the system state is not lost. If the system was powered off then the optical sensors would need to be recalibrated and the count of bad bananas processed would be lost.

Voltage Clipping Zener Diodes:

Zener diodes were used to protect the ADC chip from excess voltages coming from the angle and optical sensors via the interface board. The zener diodes are connected from the ADC channel input to ground, limiting the voltage to a range of -0.7V to 5.1V. This protects the ADC from damage and does not interfere with the ADC sampling since the sampling range of the ADC is from 0V to 5V.

Filtering Capacitors:

Capacitors were added in parallel to the internal feedback resistor of each DAC to filter noise coming out of the DACs. Since the positive terminals of the op-amps are grounded, any noise on the ground line is amplified by the op-amps when the output of the DAC is 0V. This causes significant ringing in the DAC output when the desired output is 0V. The addition of the two 10 μ F capacitors filtered out this noise.

Pushbutton:

The state of the pushbutton was determined by sampling the output of the pushbutton as an analog signal using the ADC. The pushbutton state was treated as an analog signal to reduce the number of chips required by the system. Only 2 channels of the 4-channel ADC were used on the sensors, allowing the pushbutton input to be sampled as an analog signal using a free channel without the addition of any new circuitry. If the pushbutton was latched into the system as a digital signal, an extra latch would have been required.

2.3 Software Design

2.3.1 Module Hierarchy Diagram

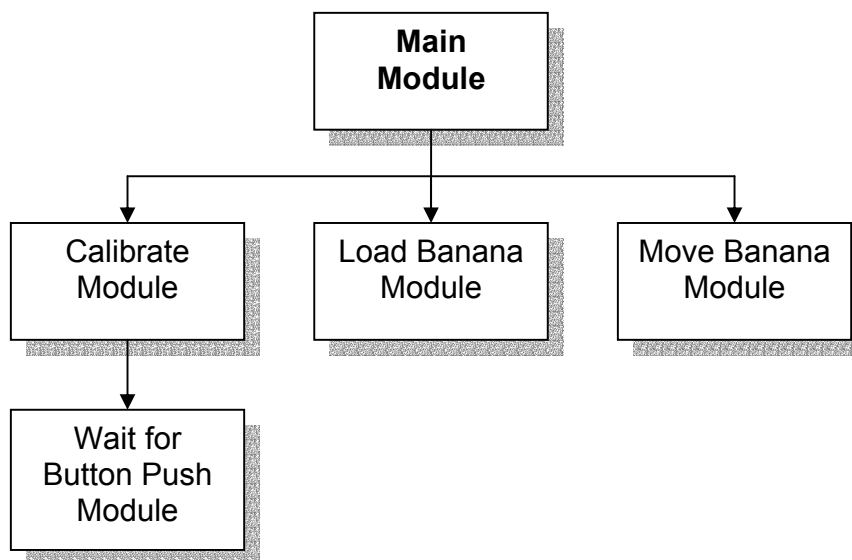


Figure 6: Module Hierarchy

2.3.2 Module Descriptions

Main Module: void main(void)

Functional description:

This module controls the system initialization and system operation. The Calibrate module is called to perform a calibration of the optical sensor input. The Load Banana module is called to load a banana under the optical sensor. The Move Banana module is called to move the conveyor belt in the forward or backward direction for a specified distance depending on the colour of the banana detected.

Inputs: None

Outputs: None

Flowchart:

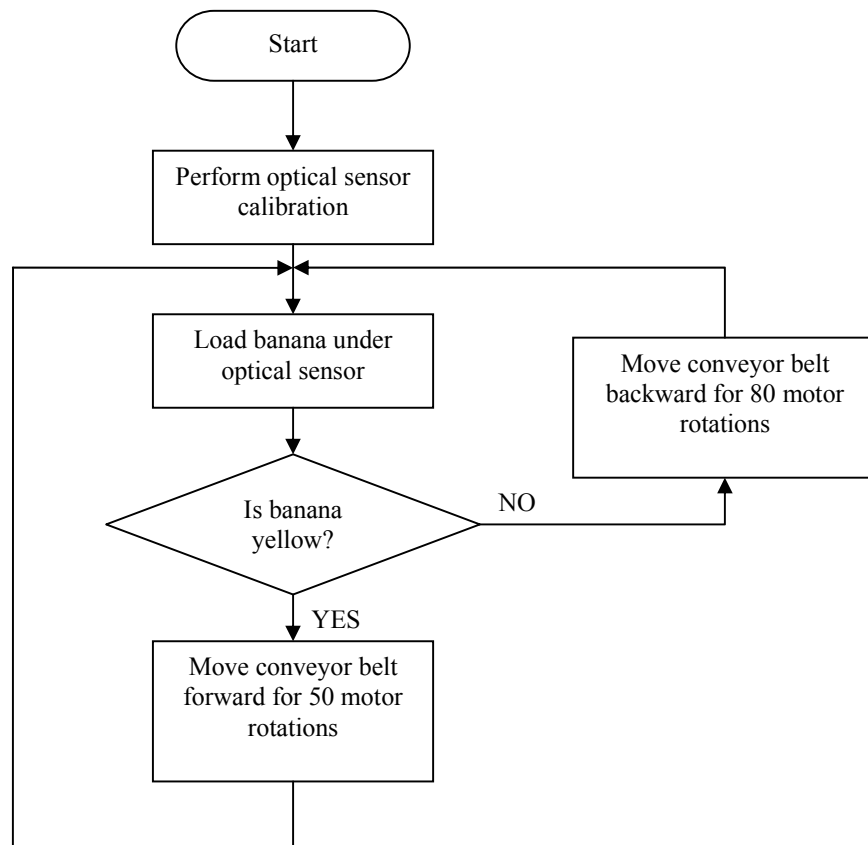


Figure 7: Flowchart of Main Module

Calibrate Module: void calibrate(void)

Functional description:

This module performs a calibration of the optical sensor input. The system is “trained” by the user to recognize the colours black, yellow, and blue. The system waits for the calibration push button to be pressed before the reading of each colour. The Wait for Push Button module is called to wait for a push of the calibration button.

Inputs: None

Outputs: Reference optical sensor values for black, yellow, and blue

Flowchart:

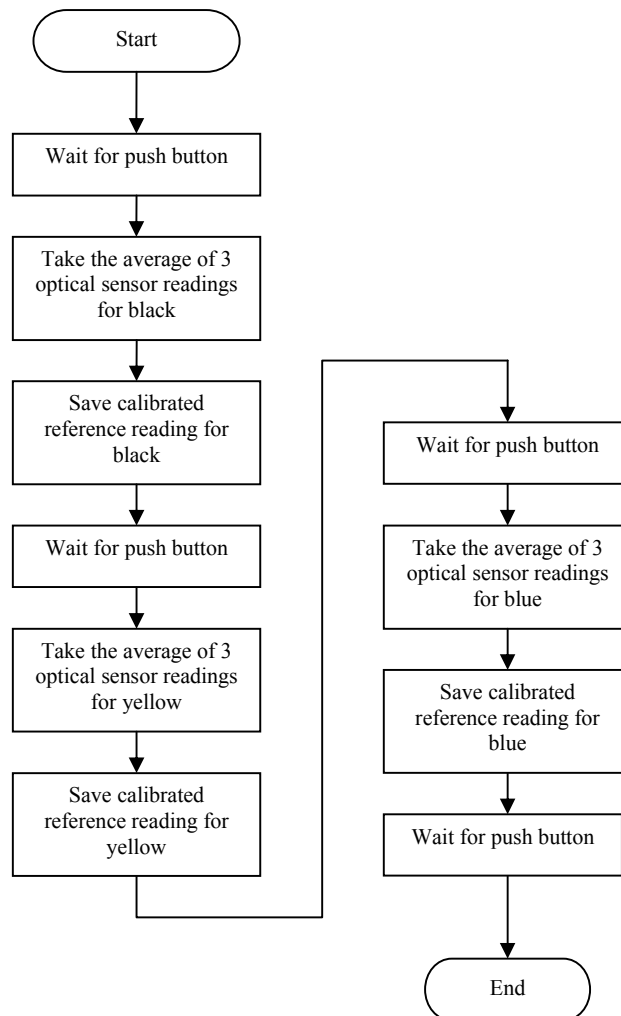


Figure 8: Flowchart of Calibrate Module

Load Banana Module: int loadBanana(void)

Functional description:

This module moves the conveyor belt forward until a banana is detected under the optical sensor. This is accomplished by activating the motor in the forward direction until a blue or yellow reading is taken from the optical sensor. When no banana is underneath the sensor, the reading should be of the black conveyor belt. When a colour is identified, the new reading is used to update the calibrated reference values of the identified colour. The new reference value is the average of the old reference value and the newly identified sensor reading.

Inputs: None

Outputs: Colour of the brick detected using the sensor
Updated the calibrated reference values for each colour

Flowchart:

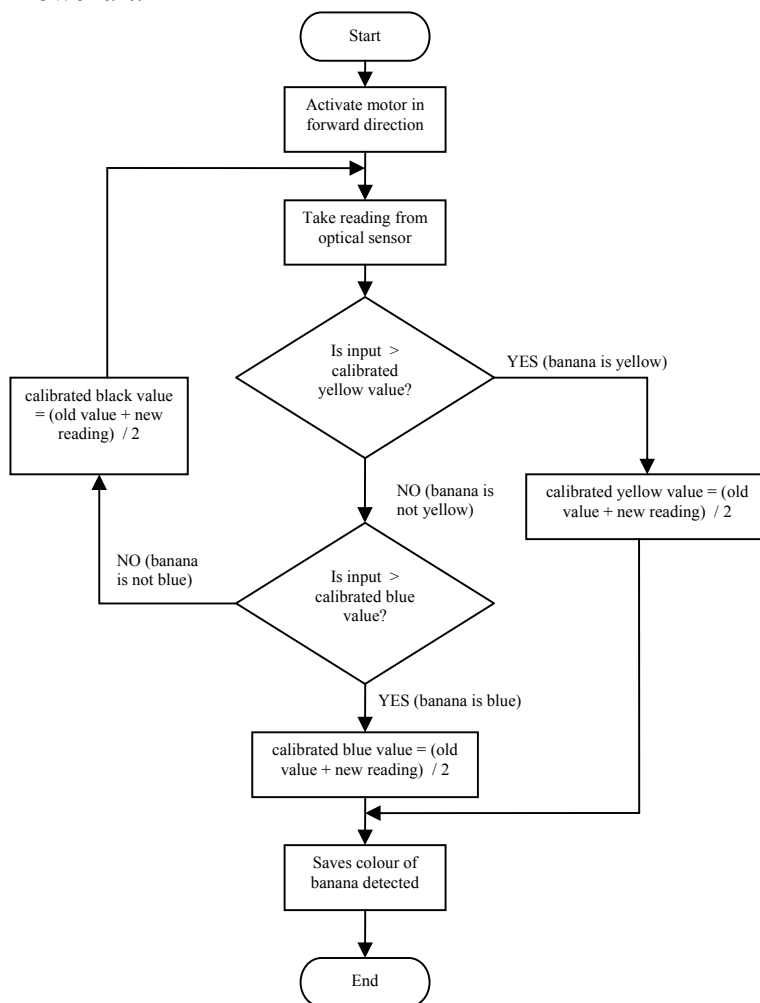


Figure 9: Flowchart of Load Banana Module

Move Banana Module: void moveBanana(int direction, unsigned short turns)

Functional description:

This module moves the conveyer belt through a set distance in a specified direction. This is performed by activating the motor in the requested direction and then deactivating the motor when the desired number of motor rotations has been counted. A motor rotation is detected by monitoring the angle sensor input.

Inputs: Number of rotations for motor to turn
 Direction for motor to turn in

Outputs: None

Flowchart:

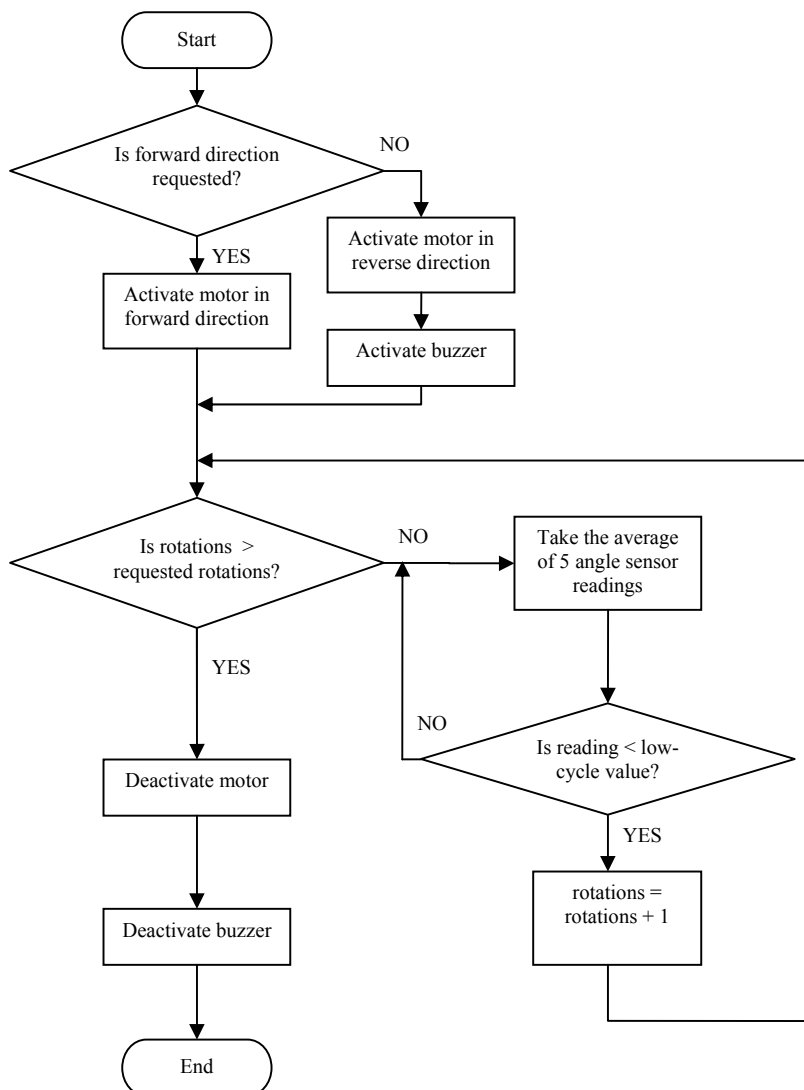


Figure 10: Flowchart of Move Banana Module

Wait for a Button Push Module: void waitForButton(void)

Functional description:

This module waits for the calibration push button to be pressed and released.

Inputs: None

Outputs: None

Flowchart:

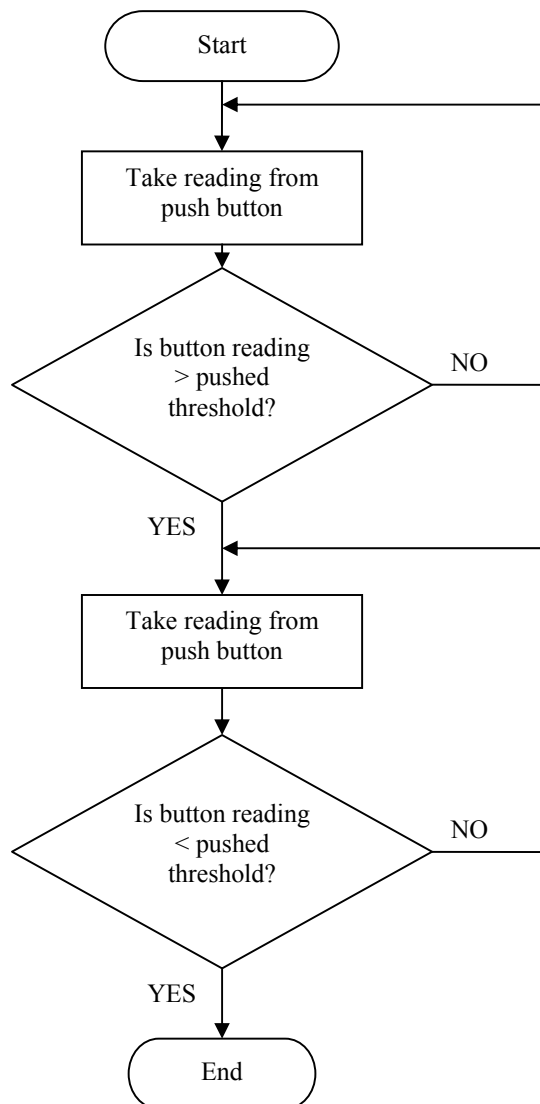


Figure 11: Flowchart of Wait for a Button Push Module

2.3.3 Software Design Decisions

Adaptive Optical Sensor Calibration:

The biggest software challenge was the interpretation of the optical sensor input. The optical sensor produces a voltage that is proportional to the amount of light received. Thus, dark-coloured objects yield a low voltage and light-coloured objects yield a high voltage. However, these voltage levels do not remain constant and drift as ambient temperature changes. To account for the change in optical sensor input between uses, a calibration stage was added to the system start-up procedure. This calibration stage “teaches” the system what voltage levels to expect for yellow, blue, and black. Once the system has started, it uses an adaptive algorithm that tracks the changes in the voltage levels for each colour. The reference voltage level for each colour is changed dynamically each time a colour is recognized. This technique allows the drifting of the optical sensor voltage to be overcome.

After the initial manual calibration of the optical sensor, no further calibrations are required as long as both good and bad bananas are processed by the system at regular intervals. Each time the system processes a banana, it “relearns” the correct voltage level for the colour of the banana processed. However, if the system is left idle for too long, then the sensor voltage level may drift enough such that the system will no longer be able to correctly identify the proper colours. At this point, the system will need to be reinitialized by pressing the system reset button and then manually recalibrated.

Angle Sensor Interpretation:

As the angle sensor rotates, it produces a sequence of voltage levels. The angle sensor input is interpreted not by monitoring the values of these voltage levels, but by detecting changes in voltage levels from one sample to the next. Since the system does not care if the angle sensor is rotating in a clockwise or counterclockwise direction, there is no need to keep track of the voltage sequence. As long as the voltage is changing, then the motor is turning. When the motor is stopped, the angle sensor voltage is constant. The angle sensor voltage tends to drift with changes in ambient temperature, but this does not present a problem since the exact voltage levels are not considered. This drift does not produce sharp changes in voltages that could be misinterpreted as motor movement.

2.3.4 Source Code

Each software module described in the previous section was implemented using C code. Please refer to Appendix A.1 for the fully-commented system source code.

2.4 Locate Implementation

2.4.1 Memory Map

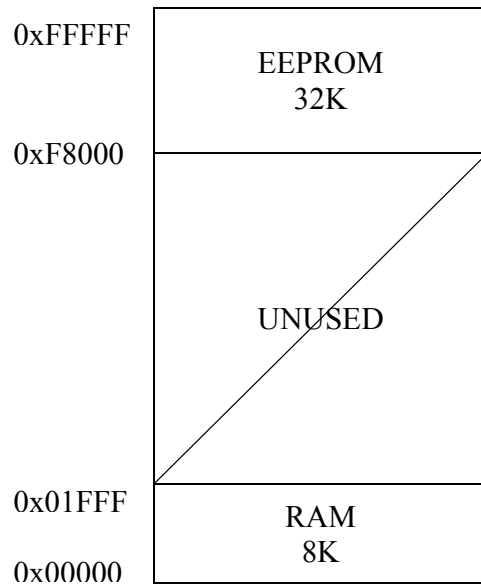


Figure 12: Memory Map

2.4.2 I/O Map

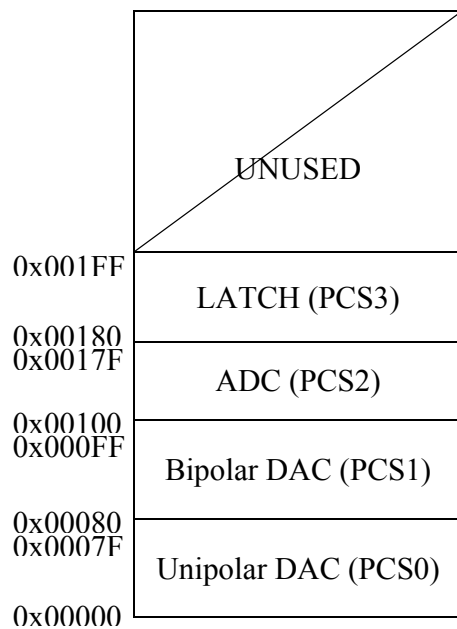


Figure 13: I/O Map

2.4.3 Configuration File

```
// Paradigm LOCATE configuration file for a Paradigm C++ 16-bit embedded
// system application.
//

cputype      I80C188XL           // Select the 80C188 as the target
                                     // system processor

hexfile      intel86
absfile      axe86
listfile     symbols            \
                segments

map 0x00000 to 0x01fff as rdwr    // RAM address space (8KB)
map 0x02000 to 0xf7fff as reserved // Unmapped address space
map 0xf8000 to 0xfffff as ronly  // EPROM address space (32KB)

initcode     reset              \ // Reset vector to program entry point
                umcs = 0x3800   \ // 32KB EPROM, 0 wait states
                lmcs = 0x01c0   \ // 8KB RAM, 0 wait states
                pacs = 0x0000   \ // PCS's start at address 0x0000
                mpcs = 0x0080   \ // PCS's mapped to I/O space

class CODE = 0xf800
class DATA = 0x0040
class ??LOCATE = 0xffff0

order DATA              \ // RAM class organization
    BSS                  \
    NVRAM                 \
    EDATA                 \
    STACK                 \
    FAR_DATA ENDFAR_DATA \
    FAR_BSS ENDFAR_BSS   \
    FAR_HEAP ENDFAR_HEAP

order CODE               \ // EPROM class organization
    INITDATA EXITDATA   \
    FAR_CONST ENDFAR_CONST \
    ROMDATA ENDROMDATA   \
    ROMFARDATA ENDROMFARDATA

output CODE             \ // Classes in the output file(s)
    INITDATA EXITDATA   \
    FAR_CONST ENDFAR_CONST \
    ROMDATA ENDROMDATA   \
    ROMFARDATA ENDROMFARDATA \
    ??LOCATE
```

2.4.4 Locate Memory Mapping

The Locate utility assigns the memory mapping for the program based on the configuration file parameters. The following is a listing of the actual assigned memory mapping:

The code, data, and stack memory regions have been highlighted in bold.

Memory Address Map for Program final

Start	Stop	Length	Segment	Class
000400H	00042DH	0002EH	_DATA	DATA
00042EH	00042EH	00000H	_CVTSEG	DATA
00042EH	00042EH	00000H	_SCNSEG	DATA
000430H	000455H	00026H	_BSS	BSS
000460H	000460H	00000H	_NVRAM	NVRAM
000460H	00046FH	00010H	_EDATA	EDATA
000470H	000470H	00000H	_NHEAP	STACK
000470H	00086FH	00400H	_STACK	STACK
000870H	00087FH	00010H	_ESTACK	STACK
000880H	000880H	00000H	_BFD	FAR_DATA
000880H	00088FH	00010H	_EFD	ENDFAR_DATA
000890H	000890H	00000H	_BFB	FAR_BSS
000890H	00089FH	00010H	_EFB	ENDFAR_BSS
0008A0H	0008A0H	00000H	_FARHEAP	FAR_HEAP
0008A0H	0008AFH	00010H	_EFH	ENDFAR_HEAP
0F8000H	0F8564H	00565H	_TEXT	CODE
0F8570H	0F857BH	0000CH	_INIT_	INITDATA
0F857CH	0F857CH	00000H	_INITEND_	INITDATA
0F8580H	0F8580H	00000H	_EXIT_	EXITDATA
0F8580H	0F858FH	00010H	_EXITEND_	EXITDATA
0F8590H	0F8590H	00000H	_BFC	FAR_CONST
0F8590H	0F859FH	00010H	_EFC	ENDFAR_CONST
0F85A0H	0F85A0H	00000H	_RD	ROMDATA
0F85A0H	0F85AFH	00010H	_ERD	ENDROMDATA
0F85B0H	0F85B0H	00000H	_BRFD	ROMFARDATA
0F85B0H	0F85BFH	00010H	_ERFD	ENDROMFARDATA
0FFF00H	0FFF20H	00021H	??CPUINIT	??LOCATE
0FFFF0H	0FFFF4H	00005H	??BOOT	(ABSOLUTE)

Entry point: FFFF:0000

Initial stack: 0040:0070

- The EEPROM code space is 0xF8000 to 0xF8564
- The RAM data space is 0x00040 to 0x00042D
- The RAM stack space is 0x000470 to 0x00086F

2.4.5 Symbol Table

The symbol table is generated by the Locate utility and shows where the user-defined functions are located in memory.

Func	F800:0101	_main
Scope	Start: 0105	Length: 0049 bytes
RegVar	SI	stopColor
Func	F800:014E	_loadBanana
Scope	Start: 0156	Length: 005E bytes
Auto	[BP-2]	input
RegVar	DI	done
RegVar	SI	stopColor
Func	F800:01B4	_calibrate
Scope	Start: 01BC	Length: 00D2 bytes
RegVar	DI	i
RegVar	SI	total
Auto	[BP-2]	input
Func	F800:028E	_waitForButton
Scope	Start: 0292	Length: 0055 bytes
RegVar	SI	buttonValue
Func	F800:02E7	_wait
Scope	Start: 02E7	Length: 0024 bytes
Parm	[BP+4]	milliseconds
Scope	Start: 02EB	Length: 0020 bytes
RegVar	SI	j
RegVar	CX	i
Parm	[BP+4]	milliseconds
Func	F800:030B	_moveBanana
Scope	Start: 030B	Length: 007E bytes
Parm	[BP+6]	turns
Parm	[BP+4]	direction
Scope	Start: 0313	Length: 0076 bytes
RegVar	DI	numTurns
RegVar	CX	input
RegVar	SI	i
Parm	[BP+4]	direction
Parm	[BP+6]	turns

3.1 Test Plan

The system was constructed and tested incrementally. Hardware components were tested individually whenever possible. This section (3.1) provides an overview of the test plan and the steps taken in the hardware validation process. The following section (3.2) presents the test programs that were written to test the hardware components.

The system consists of the following core components: CPU board, EEPROM, RAM, address demultiplexing latch. These components were wired-wrapped first to provide the most basic system functionality. A simple program that accessed each Peripheral Chip Select (PCS) address was written. To validate that the CPU executed the program, the CLKOUT and ALE signals were examined using an oscilloscope. The presence of clock pulses and ALE pulses confirmed that the CPU was executing some code. The oscilloscope was also used to verify the presence of low pulses on each PCS signal line. This verified that the code was being executed correctly and validated the proper operation of the system's core components.

Next, the seven segment display was wired into the system, along with the supporting circuitry. The components consisted of the 7-segment display, a Binary Coded Decimal decoder and an 8-bit latch to store values to be displayed. The 7-segment display was added to the system early so it could be used to display error codes and other output. The ability to provide feedback during testing accelerated the testing of subsequent components.

The system contains two 8-bit Digital-to-Analog converters. One DAC operates in unipolar mode (0V to +9V), while the other operates in bipolar mode (-9 to +9V). The output of the DAC is amplified using op-amps. First the unipolar DAC was tested. Values between 0x00 and 0xFF were written to the DAC. The DAC produces an output of 0V when it receives a value of 0x00 and an output of +9V when it receives a value of 0xFF. The correct output from the DAC was verified using an oscilloscope.

Next the Analog-to-Digital converter with a sampling range between 0V and +5V was added to the system. A sawtooth wave with amplitude of +5V was created using the unipolar DAC by repeatedly writing values between 0x00 and 0x80 (half-scale) to the DAC. This waveform was then sampled using the ADC. To confirm that the data was sampled correctly, each 8-bit sample was written to the bipolar DAC. The output of the bipolar DAC was verified on the oscilloscope to be a sawtooth wave, mirroring the output of the unipolar DAC. This verified that the ADC was operating correctly.

When the testing of main system components was completed, the board was interfaced with the analog I/O components of the conveyor system. The analog output to the conveyor system consisted of the voltage output of each DAC: one for driving the motor and one for sounding the alarm buzzer. The unipolar DAC output was used to test the

buzzer and the bipolar DAC output was used to test the motor. The analog inputs consisted of the optical sensor, angle sensor, and the calibration push button. These input signals were sampled using the ADC. The processing and interpretation of each analog input was tested.

3.2 Test Modules

PCS Test:

The PCS test was used to verify if the PCS signals that would be used for the project were working correctly. Only four PCS signals were tested because the design only needed four peripheral chip selects, one for each of the two DACs, one for the ADC, and one for the latch used for the LEDs and the 7-segment display. Using an oscilloscope probe, the PCS pins were observed one at a time while the test module was running. The expected result was to observe approximately 5V when the PCS signal was inactive and a momentary drop to 0V each time the PCS was activated.

```
#define PCS0 0x0000
#define PCS1 0x0080
#define PCS2 0x0100
#define PCS3 0x0180
#include <embedded.h>

void main(void) {
    int i;
    while(1) {
        outportb(PCS0, 0x00); // writes to PCS0
        outportb(PCS1, 0x00); // writes to PCS1
        outportb(PCS2, 0x00); // writes to PCS2
        outportb(PCS3, 0x00); // writes to PCS3
    }
}
```

Seven-Segment Display Test:

This test module was used to confirm that the seven-segment display worked properly. This was achieved by using an for-loop to display the values of 0 to 9 on the display. PCS3 was used to enable the latch to which the seven-segment display is connected. The expected result was to see the numerical values 0 to 9 with a 500ms delay between each number.

```
#define PCS3 0x0180
#include <embedded.h>

void main(void) {
    int i;
    for (i = 0; i < 10; i++) { // displays count value to seven
        outportb(PCS3, i); // ~ segment display
        wait(); // waits for 500 ms
    } // repeat when count value gets
} // ~ to 10
```

DAC Test:

The purpose of this module was to test the capability of both the unipolar DAC and the bipolar DAC. To achieve this goal, three hexadecimal values were written to each DAC with a 500ms delay between each value. The values were 0x00, 0xFF, and 0x80. The same values were written to each DAC. The oscilloscope was then used to verify that both DACs were behaving as expected. The expected result from the unipolar DAC was to see the values of 0V, +9V, and +4.5V with a 500ms delay between each. The expected result from the bipolar DAC was to see the values of -9V, +9V, and 0V with a 500ms delay between each.

```
#define PCS0 0x0000
#define PCS1 0x0080
#define PCS3 0x0180
#include <embedded.h>

void main(void) {
    while(1) {
        outportb(PCS0, 0x00); // write 0V to unipolar DAC
        outportb(PCS1, 0x00); // write -9V to bipolar DAC
        outportb(PCS3, 1); // show 1 on seven-segment
        // ~ display
        wait(); // wait for 500 ms
        outportb(PCS0, 0xFF); // write +9V to unipolar DAC
        outportb(PCS1, 0xFF); // write 0V to bipolar DAC
        outportb(PCS3, 2); // show 2 on seven-segment
        // ~ display
        wait(); // wait for 500 ms
        outportb(PCS0, 0x80); // write +4.5V to unipolar DAC
        outportb(PCS1, 0x80); // write 0V to bipolar DAC
        outportb(PCS3, 3); // show 1 on seven-segment
        // ~ display
        wait(); // wait for 500 ms
    }
}
```

ADC Test:

After the DACs were tested in the previous module, they were used to test the functionality of the ADC. This was achieved by writing a saw-tooth wave to the unipolar DAC with a voltage range of 0V to +4.5V. The analog output of the DAC was then sampled by the ADC. The ADC's digital output was written to the bipolar DAC. The expected result from the bipolar DAC was the same saw-tooth wave except with a range of -9V to 0V. The shift in voltage range is due to the fact that one DAC is unipolar and the other DAC is bipolar.

```
#define PCS0 0x0000
#define PCS1 0x0080
#define PCS2 0x0100

#include <embedded.h>

void main(void) {
    unsigned short data;
```

```

unsigned short i;

while(1){
    i = 0;
    while (i < 128) {
        outportb(PCS0, i);           // creates a sawtooth wave
        // writes i to DAC1
        i = i + 8;
        outportb(PCS2, 0x04);       // begin conversion of ADC chan#1
        data = inportb(PCS2);       // reads from ADC chan#1
        outportb(PCS1, data);       // writes sampled data to DAC2
    }
}
}

```

Pushbutton Test:

The next component to be tested was the push button. The main reason the push button was tested was to insure that there was no bouncing occurring when the button was pressed. The push button was connected to the ADC's third channel. The system waited for the button to be pressed and then released. When the button was pressed, 5V is presented to channel 3 of the ADC. The voltage from the button is 0V when not pressed. Each time the push button was pressed, the value displayed on the seven-segment display was incremented. It was expected that the value shown on the display would only be incremented once every time the push button was pressed.

```

#define PCS0 0x0000
#define PCS1 0x0080
#define PCS2 0x0100
#define PCS3 0x0180
#include <embedded.h>

void main(void) {
    unsigned short buttonValue;
    unsigned short count = 1;
    while(1) {
        outportb(PCS2, 0x06);       // begin conversion of pushbutton value
        buttonValue = inportb(PCS2); // reads sampled pushbutton value
        if (buttonValue < 0x80) {   // if pushbutton is high, button
            // ~ is pressed
            outportb(PCS2, 0x06);   // prime ADC for next conversion
            wait();                 // wait for debouncing
            do {
                // read from ADC for pushbutton value
                buttonValue = inportb(PCS2);
                // prime ADC for next conversion
                outportb(PCS2, 0x06);
                // repeats until pushbutton is released
            } while (buttonValue < 0x80);
            wait();
            count = count + 1;       // keep count of number of presses
            outportb(PCS3, count);   // writes number of presses to 7-
            // ~ segment display
        }
    }
}
}

```

Optical Sensor Test:

The optical sensor test module was used to ensure that the optical sensor was properly detecting the difference between the colours black, blue, and yellow. This was achieved by first calibrating the optical sensor by setting the reference values for black, blue, and yellow. It was expected that when no block was under the sensor, a value of 0 was displayed on the seven-segment display. If a 0 was displayed, it meant that the optical sensor was properly detecting the black colour of the conveyor belt. Blue and yellow blocks were inserted under the optical sensor randomly. It was expected that the display would show a value of 1 when a yellow block was under the sensor and a value of 2 for a blue block.

```
#define      PCS1          0x0080
#define      PCS2          0x0100
#define      PCS3          0x0180
#define      PCS4          0x0200
#include     <embedded.h>

unsigned short yellow_block;           // reference value for yellow
unsigned short blue_block;            // reference value for blue
unsigned short black_belt;            // reference value for black

void main(void) {
    unsigned short input;              // variable for ADC input
    calibrate();                       // perform optical sensor calibration
                                        // ~ to set reference values for
                                        // ~ yellow, blue, black

    while(1) {
        outportb(PCS2, 0x04);          // start ADC conversion of optical
                                        // ~ sensor
        input = inportb(PCS2);         // read optical sensor value
        if (input > yellow_block - 0x10)
            outportb(PCS3, 0x01);      // if the input is yellow, display 1 on
                                        // ~ seven-segment display
        else if (input > blue_block - 0x10)
            outportb(PCS3, 0x02);      // if the input is blue, display 2 on
                                        // ~ seven-segment display
        else if (input > black_belt - 0x10)
            outportb(PCS3, 0x00);      // if the input is black, display 0 on
                                        // ~ seven-segment display
        else
            outportb(PCS3, 0x08);      // if the input is unrecognized,
                                        // ~ display 8 on seven-segment display
        wait();
    }
}
```

Angle Sensor Test:

The angle sensor's role in the system is to provide feedback on the distance traveled by the conveyor belt. Each time a low signal is detected from the angle sensor, movement can be deduced. The seven-segment display was then incremented to indicate a full rotation has been achieved. The expected result was to see the value of the seven-segment display increment each time a rotation was completed.

```
#define PCS0 0x0000
#define PCS1 0x0080
#define PCS2 0x0100
#include <embedded.h>

void main(void) {
    while (1) {
        newInput = 0; // takes 5 readings from the angle
                    // ~sensor
        for (i = 0; i < 5; i++) {
            // begins sampling of the angle sensor
            outportb(PCS2, 0x05);
            // reads the angle sensor input
            input = input + inportb(PCS2);
        }

        input = input / 5; // calculate the average reading

        if (input < 0x1F) { // if a low cycle is detected on the
                            // ~ angle sensor, increment turn count
            numTurns++;
            // display number of turns on 7-segment
            outportb(PCS0, numTurns);
        }
    }
}
```

LED Test

The LEDs were tested to ensure that all four LEDs lit up correctly. PCS3 is used to enable the latch to which the LEDs are connected. Five hexadecimal values were written to the latch. These values were 0xF0, 0xE0, 0xC0, 0x80 and 0x00. After each value, a different number of LEDs would be lit. The expected result was to see the LEDs light up one by one.

```
#define PCS3 0x0180
#include <embedded.h>

#define NONE_ON 0xF0 // control word for lighting 0 LEDs
#define ONE_ON 0xE0 // control word for lighting 1 LED
#define TWO_ON 0xC0 // control word for lighting 2 LEDs
#define THREE_ON 0x80 // control word for lighting 3 LEDs
#define FOUR_ON 0x00 // control word for lighting 4 LEDs

void main(void) {
    while (1) {
        outportb(PCS3, NONE_ON); // turn off all LEDs
        wait(); // wait for 500 ms
        outportb(PCS3, ONE_ON); // turn on one LED
        wait(); // wait for 500 ms
        outportb(PCS3, TWO_ON); // turn on two LEDs
    }
}
```

```

wait(); // wait for 500 ms
outportb(PCS3, THREE_ON); // turn on three LEDs
wait(); // wait for 500 ms
outportb(PCS3, FOUR_ON); // turn on four LEDs
wait(); // wait for 500 ms
// repeat
}

```

3.3 Test Log

Please refer to Section 3.2 for the expected results of each test.

Module Tested	Test Shot	Date	Result	Comment
PCS Test	1	11/04/02	PCS0, PCS1, PCS2, PCS3 low pulses are observed	Basic operation of CPU, RAM, EEPROM is validated
Seven-Segment Test	1	11/05/02	No display, all segments are blank	Schematic error found, the /RBO and /RBI pins have been reversed
	2	11/05/02	Counts from 0 to 9 correctly	Seven-segment display is OK
DAC Test	1	11/06/02	Output of unipolar DAC is sinusoidal when output should be 0V Output from bipolar DAC is always +9V regardless of digital value written	Filtering capacitors were added across the feedback resistance of both DACs to reduce noise
	2	11/06/02	Output of unipolar DAC is noise-free and produces expected voltages Output from bipolar DAC is unchanged	Unipolar DAC is OK Added 10K resistor to the output of bipolar DAC's op-amp circuit to load the DAC
	3	11/08/02	Output of bipolar DAC varies with the digital input, but voltages are in the range of -6V to +6V instead of -9V to +9V for a while, then DAC stopped working	DAC and op-amp were damaged in test and replaced Schematic error was found: the 2in+ and 2in- pins of the op-amp amplifying bipolar DAC was reversed 10K resistor from previous test was removed
	4	11/08/02	Output of bipolar DAC varies from -9V to +9V correctly	Bipolar DAC is OK

ADC Test	1	11/12/02	ADC sampled the output of the unipolar DAC and wrote sampled data to bipolar DAC momentarily, but then stopped working	ADC was damaged because the output of the unipolar DAC went up to +9V and ADC only operates in the 0V to +5V range Decreased output of DAC to 0V to +4.5V
	2	11/12/02	ADC sampling of unipolar DAC was too slow because of excessive software delay, making it hard to verify that the output of the bipolar DAC matched the output of the unipolar DAC	Software delay was decreased from 9 seconds to 0.5 seconds
	3	11/12/02	The output of bipolar DAC matched the output of the unipolar DAC except for the expected DC offset	ADC is OK
Pushbutton Test	1	11/13/02	One button press is interpreted as many presses, as 7-segment display counter increments many times	Increased debouncing delay in software
	2	11/13/02	No result from button press	Suspect that pushbutton is faulty, and was replaced
	3	11/13/02	Each button press increments 7-segment display counter by one	Pushbutton is OK
Optical Sensor Test	1	11/12/02	Unrecognized data values were sampled by ADC, no colour recognition observed	Interface board potentiometer for optical sensor adjusted to bring optical sensor output into 0V to 5V range
	2	11/12/02	Yellow is distinguished from other colours, but random results are displayed for other colours	Threshold voltage for each colour was adjusted
	3	11/12/02	Black is not recognized, an 8 is displayed when no banana is under the sensor indicating an unrecognized colour	Threshold voltage for black was increased from 0.5V to 1.0V.
	4	11/13/02	Black is distinguished from yellow and blue, but yellow is often misrecognized as blue	Margin of error for detection of yellow was increased from 0.5V to 0.8V

	5	11/13/02	Blue and yellow can be recognized properly, but black is recognized as blue	Margin of error for detection of black was decreased from 0.8V to 0.3V
	6	11/13/02	All colours are recognized	Optical sensor appeared to be OK
	7	11/17/02	All colours are recognized, but the optical sensor outputs for each colour are different than tested on 11/13/02	Calibration procedure was added to allow the reference readings for each colour to be recorded into the software upon system start-up
	8	11/17/02	All colours are recognized properly and between system start-ups	Optical sensor is OK
Angle Sensor Test	1	11/14/02	Number of rotation is counted wildly, the 7-segment display changed too fast for reading	ADC was damaged due to high voltage from the angle sensor (-6V to +9V) Zener diodes were added to the angle sensor and optical sensor input channels of ADC for protection
	2	11/14/02	Number of rotations are not counted, counted display does not change	Interface board potentiometer for angle sensor was adjusted to bring input into 0V to +5V range of ADC
	3	11/15/02	Some rotations are counted, some are not counted	Voltage threshold for detecting a rotation cycle was decreased to account for variations in angle sensor output
	4	11/18/02	Rotations of angle sensor when turned by hand is counted correctly	Angle sensor appears to be OK
	5	11/19/02	Rotations of angle sensor when driven by the motor are not counted properly	The motor turns the angle sensor much faster than previously tests performed by hand The software delay for sampling from angle sensor was reduced
	6	11/19/02	Rotations of angle sensor is correctly counted	Angle sensor is OK
LED Test	1	11/16/02	LEDs turned on one by one with small delay	LEDs are OK

3.4 Acceptance Test

The acceptance test will be used to demonstrate the proper operation of the developed system and validate that customer requirements have been met. The following test procedure will be used:

1. Connect all power connectors to the board from the DC power supply: +5V, GND, +12V, -12V.
2. Connect analog I/O of the system to the interface board using a 20-pin ribbon cable.
3. Connect LEGO connectors of the system for the angle sensor, optical sensor, motor and buzzer to the connectors on the interface board.
4. Power on the system and ensure proper system start-up. The 4 calibration LEDs should be off and the 7-segment display showing the count of bad bananas processed should display 0.
5. Perform the calibration procedure described in the User's Manual. After each step, the calibration progress bar should light an additional LED. After the calibration has been completed, all four LEDs should be on. At this point, the motor should be running.
6. Load a yellow banana into the banana loading bay. The conveyor should carry the banana forward to beneath the sensor and then continue carrying the banana forward to the banana collection bay.
7. Load a blue banana into the banana loading bay. The conveyor should carry the banana forward to beneath the sensor and then reverse direction and carry the banana backward to the banana disposal bay. The alarm buzzer should sound while the conveyor is moving in the reverse direction. The 7-segment display should show a value of 1.
8. Load random bananas into the banana loading bay. The bananas should be carried to the proper location. After each blue banana, the 7-segment display should increment in value by 1.
9. Continue loading bananas into the system for 5 minutes to demonstrate the system's use of adaptive software calibration of optical sensor input to deal with drift of optical sensor input voltages.
10. Engage the emergency override while a banana is being carried in the forward direction. The motor should stop immediately. When the emergency override is disengaged, the conveyor system should resume moving in the forward direction.
11. Engage the emergency override while a banana is being carried in the reverse direction. The motor should stop immediately. When the emergency override is disengaged, the conveyor system should resume moving in the reverse direction until the banana has been discarded. This demonstrates the proper interpretation of angle sensor input in monitoring the distance the motor has travelled.
12. Continue performing tests requested by the customer until the customer is satisfied that requirements have been met and the system is functioning correctly.

4.1 Final Design Recommendations

The optical and angle sensors provided are not robust enough for the industrial setting. Although software techniques have been applied to overcome some of the shortcomings of these sensors, it is recommended that more reliable sensors be used when the system is implemented. In particular, the sensors should be more tolerant to ambient temperature changes. Also, the optical sensor should be more tolerant of changes in ambient light levels. A more robust optical sensor would reduced the system overhead required for sensor calibration and tracking of sensor drift.

More displays could be added to the system to provide more comprehensive feedback to the user. Currently, the system displays a count of bad bananas processed using a single digit 7-segment display. A double digit display would increase the range of the counter before a roll-over occurs. Also, additional displays could be used to show the number of good bananas processed or the good banana to bad banana ratio.

The system could be expanded to recognize other colours. For example, not all bananas are blue or yellow. Some bananas have been known to turn black during the freezing process. Although it would be difficult to differentiate black bananas from the black conveyor belt, this could be achieved with better optical sensors. In addition, the system could be altered with relative ease to process completely different fruits and vegetables such as green cucumbers, purple eggplants, or orange oranges.

4.2 Project Status

All specified functionality has been achieved and validated. The final system meets or exceeds all original requirements. In particular, the unreliable nature of the provided optical sensor was countered using adaptive software techniques for colour calibration and colour tracking. Accuracy of banana processing is guaranteed to be within the specified error tolerance margins. Development and validation of the system was completed within the contracted time period. The system is ready for acceptance testing.