University of Florida
Department of Electrical and Computer Engineering
EEL 5666
Intelligent Machines Design Laboratory
Spring 2005

Special Sensor Report

Instructor: Dr. A. A. Arroyo

TAs: William Dubel and Steven Pickles

Student: Albert Chung

Date: 03/14/2005

Table of Contents

I. Abstract

The main purpose of the Automated Storage and Retrieval System is to coordinate with an Automated Guided Vehicle to seamlessly and efficiently transport pallets on and off a shelf, where they will arrive at an outgoing dock. To achieve this goal, some type of wireless communication must be utilized. I have chosen radio frequency communication using a Laipac TRF-2.4G module.

II. Special Sensor

i. Introduction

The Laipac TRF-2.4G transceiver uses a Nordic nRF2401 VLSI chip with a 16MHz crystal oscillator and a built in dipole antenna. The benefits of the Laipac transceiver over a standard RF receiver/transmitter pair are the single chip/device operation for bidirectional communication, dual channel operation, hardware Cyclic Redundancy Checksum (CRC) code generation and error checking, and high speed ShockBurst transmission. The latter three are useful for microcontroller application since they free up processing power required to implement a lower level datalink protocol. With that many features, you would expect the device to cost a fortune; however, I purchased two modules from Spark Fun Electronics (www.sparkfun.com) for $19.95 each along with two breakout boards that were $0.95 a piece. Unless you plan on designing your own board that houses all of your circuitry, I recommend buying the board for the extra small fee since the transceiver does not utilize standard 0.1" pin spacing.

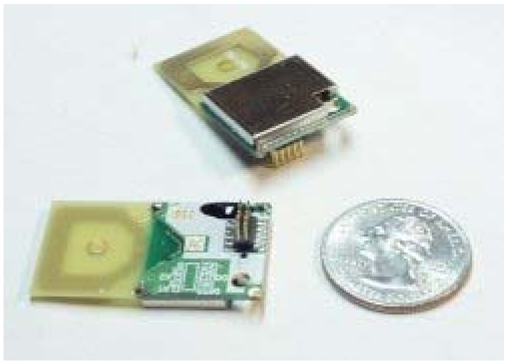


Figure 1. Physical Device Features

ii. Operation

The nRF2401 is a low voltage, CMOS IC that requires a source voltage, Vcc, and high logic levels equal to 3.3 ± 0.3 V. In order to interface the module with a TTL logic device, the device outputs must be passed through a TTL to 3.3V level converter. A suitable device is a MAX3001E level translator by Dallas Semiconductor (www.maxim-ic.com). These can be obtained for free through their sampling program; however, the devices only come in surface mount packages. Due to the time constraints for this project I was unable to send off a board design to mount the IC in time for completion. Instead, I switched to a 3.3V system since my PIC 18F8720, analog sensors, motor drivers, and lcd all support low power operation.

The transceiver can operate in two modes: ShockBurst and Direct Mode Tx/Rx. In ShockBurst mode the device handles the preamble and CRC, and transmits the data at a high data rate of 250kpbs or 1Mbps (set by the user). When a packet is received, the device will notify the microcontroller by setting a flag, and the data will remain in a buffer until the microcontroller clocks it out serially. In direct mode, the transceiver acts as a modulator that modulates an incoming data stream onto a 2400MHz carrier. The microcontroller would have to create a preamble, monitor the airwaves for a valid packet, and handle error checking in software. I opted to operate the device in ShockBurst mode to allow the microcontroller to process other tasks.

a. Configuration

The TRF-2.4G is configured with a 120 bit configuration word that sets the data length, address, address length, CRC, two channel mode, device mode, data rate, operating frequency, crystal frequency, and Rx or Tx operation. Sending data to the transceiver's buffer is fairly simple. The configuration word is sent to the device serially (MSB first) to the DATA I/O port on the device and is clocked in by the microcontroller on successive rising clock edges. Pay close attention to the timing requirements specified by the device in the product datasheet. A setup time of 500ns is required for the data to be on the bus before a rising clock edge is triggered. Following, the clock and the data must be held for 500ns before the clock edge falls.

To enter configuration mode, set CE and CS low through an output port on the microcontroller. This sets the device in standby and selects the configuration register to be written to. A subsequent 5μs delay is required before sending data to the device. I configured the transceiver to operate on a single 2500MHz channel in ShockBurst mode operating at 250kbps, 0dB transmission, with an 8-bit address, 16-bit data, and 16-bit CRC length. The configuration word is shown below using an arbitrary address and initialize in RX mode. Once the configuration word is fully sent, the configuration register in the transceiver is updated when CS is set high. The device only updates the number of bits actually sent, which is helpful because during receiving and transmission, the last bit, RXEN, needs to be set or disabled to switch between receive and transmit mode.

Channel 2 Data Width							
119	118	117	116	115	114	113	112
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Channel 1 Data Width							
111	110	109	108	107	106	105	104
0	**0**	**0**	**1**	**0**	**0**	**0**	**0**

Channel 2 Address (8 to 40 bits)																							
103	102	101	100	99	98	97	96		95	94	…	73	72		71	70	69	68	67	66	65	64
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**		**0**	**0**	**…**	**0**	**0**		**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Channel 1 Address (8 to 40 bits)																							
63	62	61	60	59	58	57	56		55	54	…	33	32		31	30	29	28	27	26	25	24
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**		**0**	**0**	**…**	**0**	**0**		**1**	**1**	**0**	**1**	**1**	**1**	**0**	**0**

Address Width						**CRC Length**	**CRC Enable**
23	22	21	20	19	18	17	16
0	**0**	**1**	**0**	**0**	**0**	**1**	**1**

2 Channel Enable	**ShockBurst**	**RF Data Rate**	**Oscillator Freq**			**RF Power**	
15	14	13	12	11	10	9	8
0	**1**	**0**	**0**	**1**	**1**	**1**	**1**

Channel Frequency							**RXEN**
7	6	5	4	3	2	1	0
1	**1**	**0**	**0**	**1**	**0**	**0**	**1**

Table 1. ASRS Configuration Word

b. Transmission

To begin transmitting data, the device must be set to TX mode in the configuration register (RXEN = 0) and be brought out of standby by setting CE high. Also, the data register must be selected by setting CS low. In ShockBurst mode transmission the device receives a data packet that includes a destination address and a payload from the microcontroller. The packet is sent serially (MSB first) and is clocked into a buffer by the microcontroller in the same manner as sending the configuration. When the full packet is sent to the device, setting the CE pin low will activate the onboard processing, which includes appending a preamble and CRC checksum, and the transceiver will begin broadcasting the data wirelessly.

c. Receiving

To begin receiving data, the device must be set to RX mode in the configuration register (RXEN = 1) and be set to active mode by setting CE high. The transceiver will monitor the airwaves for a valid packet by locking onto the preamble and checking the address and CRC checksum. If the address and CRC are valid, the packet is stripped of the overhead and the data is placed into a buffer for the microcontroller to clock out. The transceiver will set the data ready pin high to notify the microcontroller, which can be either polled or tied to an external interrupt pin. When the data is clocked out and the buffer is empty, the data ready flag will be set low again and the transceiver will begin monitoring for other packets.

iii. Implementation

I sent data to the transceiver by using basic I/O ports to connect the CE, CS, CLK, and DATA lines. Serial transmission was achieved by shifting an array of 8 bit words onto a single I/O pin tied to the DATA pin on the TRF-2.4G and then toggling the CLK pin. I set the device to continuously monitor for an incoming packet unless it has data to send, and used an interrupt call a function to clock out the received values. I will probably change to the polling method in the final design, since the main routine needs to acknowledge that incoming data has arrived anyway. I will also implement a stop and wait protocol and add beginning and end headers to the frame to ensure that no information is lost between transceivers.

iv. Experiments

I wrote a small routine to increment a counter and send it to another microcontroller without additional error checking protocols. The number of correctly and incorrectly received values were recorded for varying distances[1]. The results are shown below in table 2. Even at short range operation, the system integrity isn't very high. Several packets were lost during transmission and some random values were also received. This is likely due to the noise in the communication channel caused by several other household devices (802.11, microwaves, cordless phones, etc) operating at the same frequency. The results confirm that a software protocol must be implemented to retransmit lost packets.

[1] Due to the pressures of a massive Communications exam, the experiment was only conducted for a single trial. Hence, the results show random phenomenon without a distinct pattern. A better experiment would be do several trials, and take the average of each trial.

Distance (ft)	Number of Missed Packets	Number of Correctly Sent Packets	% Error
3	2	254	0.78125
5	7	249	2.734375
7	2	254	0.78125
9	7	249	2.734375
11	4	252	1.5625

Table 2. Experiment Results

III. Accompanying Sensors

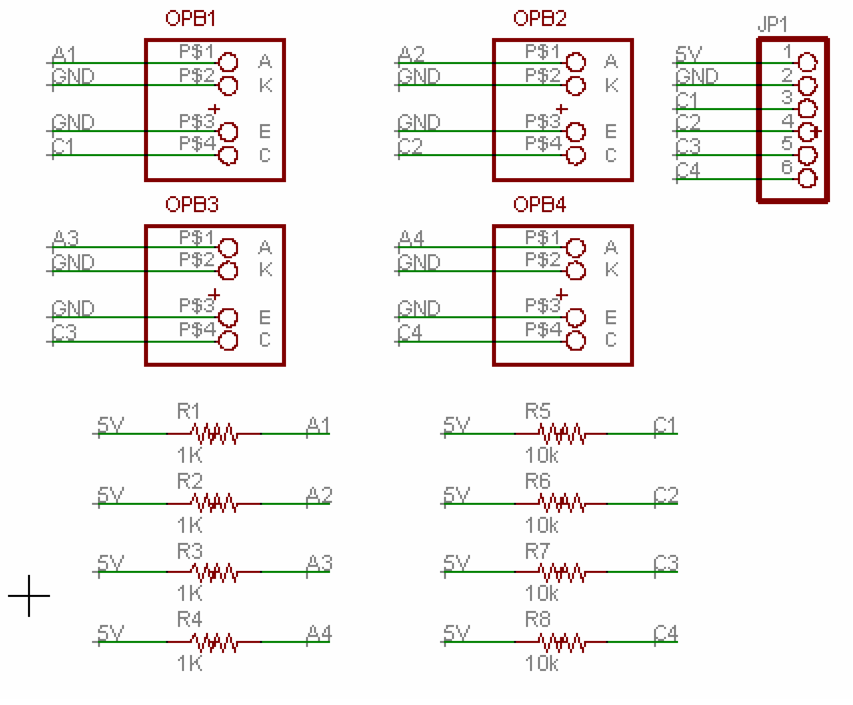
i. Bump

Rear bump sensors serve several primitive functions on the ASRS. Two switches were placed on the rear bumper that triggers a routine to disable the motors and stop the vehicle when activated. I wired the switches in parallel between ground and a pull up resistor to 3.3V in order to save an I/O pin on my microcontroller. This is suitable for my purposes since the direction that the bump occurs is not important since the ASRS is a line follower.

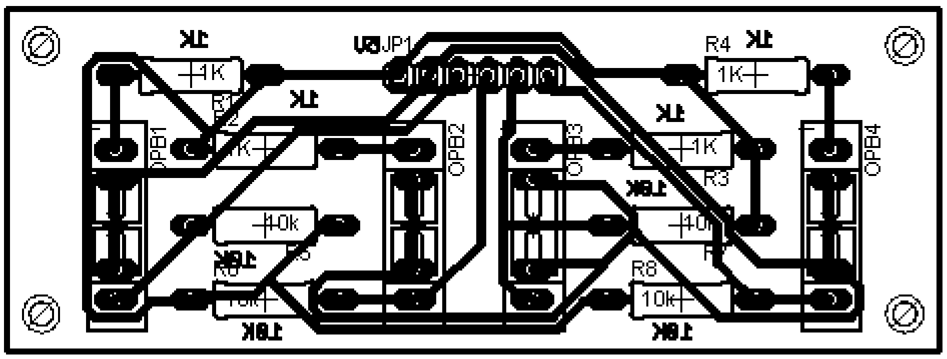
Bump switches were also used to track the vertical position of the fork. I placed two lever switches to serve as limit switches for the fork at the top and bottom of the

ii. Line Following

I'm using four OPB745 IR emitter/detector pairs to track a line. I found that the lowest price for these sensors were at Mouser Electronics (www.mouser.com) for $3.60 per unit. The OPB745 is composed of an IR LED and a bipolar junction phototransistor whose output is controlled by incoming infrared light entering the base terminal. The schematic for the circuit and a layout example is shown below. I observed that that under indoor lighting, a solid white surface would produce a converted analog value around 135 and a black electrical tape surface would produce a value above 230. The best orientation of the device is at around a 15º angle above the surface pivoted on the phototransistor. About an eighth of an inch clearance provides enough space between the lens and the surface to produce accurate results.



a.



b.

Figure 2. a. Line Follower Schematic b. PCB Layout

iii. IR Proximity

Two short-range Sharp GPD2D120 IR distance sensors will provide forward path vision. By far, the cheapest place to purchase the sensors is from the Mark III Robot Store (www.junun.org) for $8.25 each and a cable can be purchased for an extra $1.10. I've had good experience ordering from them and received my order in only a few days. The only downside is that no order status or tracking number is provided.

The choice of the short range over the long range GPD2D12 sensors is due to the tight spaces that the ASRS will be operating in. I didn't want the robot to be constantly detecting objects in the distance and I also required it to approach a shelf close enough to time the rest of the way there with little error. With the two sensors facing forward and crossing paths, I concluded that a value of 80 translated to around 4cm (minimum distance detection) and anything above 50 should represent an obstacle fairly close to the ASRS.

IV. Sources for Parts

Laipac TRF-2.4G RF Wireless Transceivers:

www.sparkfun.com

Price: $19.95 each (+$0.95 for a breakout board)

MAX3001E Level Translators:

www.maxim-ic.com

Price: Free

Push Button Switches:

IMDL Lab

Price: Free

Roller Lever Switches:

Radio Shack

Price: $2.50 each (cheaper at Jameco ~ $1.25)

OPB745 IR Emitter/Detector Pairs:

www.mouser.com

Price $3.60 each

Sharp GPD2D12 Distance Measuring Sensors:

www.junun.org

Price: $8.25 each (+$1.10 for a cable)

V. References

Nordic Semiconductor nRF2401 Datasheet:

http://www.sparkfun.com/datasheets/RF/nRF2401rev1_1.pdf

Laipac TRF-2.4G Datasheet:

http://www.sparkfun.com/datasheets/RF/RF-24G_datasheet.pdf

William Dubel's Reliable Line Tracking Report:

http://www.mil.ufl.edu/imdl/handouts/lt.doc

VI. Appendix

Sample Code

```
/********************************************************
 * RF LINK                                                 *
 *                                                         *
 * Written for the PIC18F8720 @ 20 MHz (5MHz Instruction)  *
 * Interfaces: Digital I/O on RH0, RH1, RH2, RH3,          *
 *        and INT1, that connect to a TRF-24G RF module    *
 * Copyright (2005) Albert Chung                           *
 *                                                         *
 * NOTES:                                                  *
 * Hardware Connections:                                   *
 *         RH0 (I/O):      TRF-24G Data                    *
 *        H1 (Output):     TRF-24G CLK1                    *
 *        H2 (Output):     TRF-24G CS                      *
 *        H3 (Output):     TRF-24G CE                      *
 *        NT1 (I/O):       TRF-24G DR1                     *
 ********************************************************

#include <p18f8720.h>
#include <delays.h>
#include "interrupts.h"

#define DATA PORTHbits.RH0
#define CLK1 PORTHbits.RH1
#define CS PORTHbits.RH2
#define CE PORTHbits.RH3

#define asrs_addr 0b11011101
#define agv_addr 0b11100110
#define data_w 16
#define addr_w 8
#define crc 0b11                 // CRC enable
#define mode 0b01001111          // Rx2En = 0,  Shockburst Mode, 250kbps,16 MHz module crystal,  0db Power
#define rf_ch 0x64                      // 2500 MHz frequency channel
#define ACK 0x97D3               // Acknowledgement for stop and wait protocol (not used yet)

int payload = 0;

void CLK (void)
{
        Delay10TCYx(10);                // 500 nsec (tsetup)
        CLK1 = 1;                       // clock in the value
        Delay10TCYx(10);                // 500 nsec (thold)
        CLK1 = 0;
}

void Tx_En(void)                 // Set module to active transmit mode
{
        int i;
        char Tx_2500MHz = (rf_ch << 1);

        DDRHbits.RH0 = 0;
        DDRHbits.RH1 = 0;
        DDRHbits.RH2 = 0;
        DDRHbits.RH3 = 0;

        CE = 0;                         // Set configuration mode
        CS = 1;                         // Select configuration register

        Delay10TCYx(40);                // 10 usec (tcs2data)

        for ( i = 7; i >= 0; i--)
        {
                DATA = Tx_2500MHz >> i;                 // Shift out Rx address (MSB first)
                CLK();                                  // Clock in the data
        }

        CS = 0;                         // shift the configuration word into the module

        RF_IRQ_OFF;                     // Disallow RD1 to interrupt MCU

        Delay100TCYx(100);              // 250 usec (tsettling)

        CE = 1;                         // Turn on ACTIVE TX Mode

        Delay10TCYx(40);                // 10 usec (tce2data)
}

void Rx_En(void)                        // Set module to active receive mode
{
        int i;
        char Rx_2500MHz = (rf_ch << 1) | 1;             // Last byte in configuration word

        DDRHbits.RH0 = 0;               // Set Outputs
        DDRHbits.RH1 = 0;
        DDRHbits.RH2 = 0;
        DDRHbits.RH3 = 0;

        CE = 0;                         // Set configuration mode
        CS = 1;                         // Select configuration register

        Delay10TCYx(40);                // 10 usec (tcs2data)

        for ( i = 7; i >= 0; i--)
        {
                DATA = Rx_2500MHz >> i;                 // Shift out Rx address (MSB first)
                CLK();                                  // Clock in the data
        }


        CS = 0;                         // shift the configuration word into the module

        DDRHbits.RH0 = 1;               // Set Data as Input

        Delay100TCYx(100);              // 250 usec (tsettling)

        RF_IRQ_ON;                      // Allow RD1 to interrupt MCU

        CE = 1;                         // Turn on ACTIVE TX Mode
        Delay10TCYx(40);                // 10 usec (tce2data)
}

void Transmit (int tx_payload)
{
        int i;

        Tx_En();

        for ( i = 7; i >= 0; i--)
        {
                DATA = agv_addr >> i;                   // Shift out Rx address (MSB first)
                CLK();                                  // Clock in the data
        }

        for ( i = 15; i >=  0; i--)
        {
                DATA = tx_payload >> i;                 // Shift out payload (MSB first)
                CLK();                                  // Clock in the data
        }

        CE = 0;                                         // Activate Shockburst Tx

        Rx_En();
}

void Receive (void)
{
        int i;
        payload = 0;

        for ( i = 15; i >=  0; i--)
        {
                CLK1 = 1;                               // Clock in the data
                Delay10TCYx(10);                        // 500 nsec (tsetup)
                payload |=  DATA << i;                  // Shift in payload (MSB first)
                CLK1 = 0;
                Delay10TCYx(10);                        // 500 nsec (tsetup)
        }

//      Transmit(ACK);                                  // Send acknowledgement
}

void Init_RF (void)
{

// Set up the configuration packet in segments of 8 bits
// {data_w, addr2 not used = 5x"0", redundant address bits exceeding address width = 3x"0", asrs_addr, addr_w[bit7:2]
crc[bit1:0], mode, rf_ch & receive mode}

        char addr_w_crc = ( addr_w << 2 ) | crc;
        char config[15] = {0,data_w, 0, 0, 0, 0, 0, 0, 0, 0, asrs_addr, addr_w_crc, mode, (rf_ch << 1)};
        int i, j;

        DDRHbits.RH0 = 0;
        DDRHbits.RH1 = 0;
        DDRHbits.RH2 = 0;
        DDRHbits.RH3 = 0;

        Delay10KTCYx(10);       // 5 msec (tpd2sby)

        CE = 0;                         // Set configuration mode
        CS = 1;                         // Select configuration register

        Delay10TCYx(40);        // 10 usec (tcs2data)

        for ( i = 0; i < 15; i++)
        {
                for ( j = 7; j >= 0; j--)               // send the configuration word MSB first
                {
                        DATA = config[i] >> j;          // shift config word 1 bit at a time
                        CLK();                                  // Clock in the data
                }
        }

        CS = 0;                         // Shift the configuration word into the module

        Rx_En();                        // Set the module to active Rx mode
}

void Stop_And_Wait (void)
{
        //…
}
```