RFM Bit Level Radio for Sensor Networks

CS 603: Sensor Networks
Dr. Gupta
February 5, 2004
Presented by:
Eric DeKoeckkiek

The RFM TR1000 Low Power Radio Transceiver For the Mica Mote
- 916 Mhz Operation
- 115 Kbps in Amplitude-Shift Keyed modulation mode
- The hardware pins CNTRL0 and CNTRL1 are the main bits for handling TX, RX, and power control functionality.

Mode | CNTRL0 | CNTRL1
--- | --- | ---
Receive | HIGH | HIGH
ASK TX | LOW | HIGH
OOK TX | HIGH | LOW
Sleep | LOW | LOW

RFM Bit Level Radio for Sensor Networks

/usr/tinyos/tos/interfaces/Radio.nc
Radio.nc defines the generic interface functions used by various platforms.
This code outlines the functions defined to set the modes and bit level handling of the radio device (either the TR1000 or other devices).
TinyOS allows for hardware abstraction beginning at even the lower levels of code.
Example code pieces:

* Transition into transmit mode:
  * @return SUCCESS always
  * commandmode(TRANSMIT);
* Transition into receive mode:
  * @return SUCCESS always
  * commandmode(RECEIVE);

/*
 * Set bit rate to 0 (20 Khz), 1 (13 Khz) or 2 (10 Khz).
 * @return SUCCESS if valid setting, FAIL otherwise.
 *
 * commandsetbitrate(n); /* n = bit rate level */
*/

RFM Bit Level Radio for Sensor Networks

Example code pieces:

* Transition into transmit mode:
  * @return SUCCESS always
  * commandmode(TRANSMIT);
* Transition into receive mode:
  * @return SUCCESS always
  * commandmode(RECEIVE);

/*
 * Set bit rate to 0 (20 Khz), 1 (13 Khz) or 2 (10 Khz).
 * @return SUCCESS if valid setting, FAIL otherwise.
 *
 * commandsetbitrate(n); /* n = bit rate level */
*/

RFM Bit Level Radio for Sensor Networks

RFM.comp Component Definition

Provides the specific definition for RFM component on Mica mote.

Provides the specific interface implementation, complete with RFM.c driver implementation.

The interface of the RFM.comp is modeled closely to Radio.nc interface to further facilitate interoperability.

* This module controls the RFM radio. It accepts bits from the BYTE layer and
  * sends them to the RFM (after an appropriate delay to set the bitrate)
  * and in signals a bit continuation to complete.

*
RFM Bit Level Radio for Sensor Networks

RFM comp Declaration

Example code pieces:

```c
TOS_MODULE RFM;

ACCEPTS {
    char RFM_INIT(void);
    char RFM_TX_MODE(void);
    char RFM_TX_BIT(char data);
    char RFM_RX_MODE(void);
    char RFM_PWR(char mode);
    char RFM_SET_BIT_RATE(char level);
};

SIGNALS {
    char RFM_TX_BIT_EVENT(void);
    char RFM_RX_BIT_EVENT(char data);
};

INTERNAL {
    void SetReceiveMode(void);
    void SetTransmitMode(void);
};
```

RFM Bit Level Radio for Sensor Networks

RFM.c Header Comment

RFM.c is the actual C implementation code/driver for the radio device

```c
/*
 * This component performs bit level control over the RF Monolitics radio.
 * Additionally, it control the amount of time per bit by using TCNT1.
 * The sample period can be set to 1/2x, 3/4x, and x. Where is the
 * bit transmission period. 1/2 and 3/4 are provided to do sampling
 * and different at the gram level will be done for example.
 */
```

RFM Bit Level Radio for Sensor Networks

RFM.c Example code pieces

```c
/* This command sets the RFM component (radio) to transmit bit “data” */
char TOS_COMMAND(RFM_TX_BIT)(char data){
    //if not in the transmit mode fail.
    if(VAR(state) != 1) return 0;
    //sent the output pin accordingly.
    if(data & 0x01){
        SET_RFM_TXD_PIN();
    } else{
        CLR_RFM_TXD_PIN();
    }
    return 1;
}
```
/* This command sets the RFM component (radio) into different power modes */
char TOS_COMMAND(RFM_PWR)(char mode){
    if(mode == 0){
        // turn off the RFM chip.
        CLR_RFM_CTL0_PIN();
        CLR_RFM_CTL1_PIN();
        // disable timer1 interrupt
        outp(0x00, TCCR1B); // scale the counter
        cbi (TIMSK, OCIE1A);
        // record the current state.
        VAR(state) = 2;
        return 1;
    }else if(mode == 1){
        VAR(state) = 3;
        outp(0x09, TCCR1B); // scale the counter
        sbi (TIMSK, OCIE1A);
    }else if (mode == 2){
        // turn off the RFM chip
        CLR_RFM_CTL0_PIN();
        CLR_RFM_CTL1_PIN();
    }

    return 1;
}

/* This command sets the RFM component (radio) into transmit mode */
char TOS_COMMAND(RFM_TX_MODE)(){
    if(VAR(state) == 2) return 0;
    // set the RFM chip to TX mode.
    SET_RFM_CTL0_PIN();
    CLR_RFM_CTL1_PIN();
    dbg(DBG_RADIO, (“RADIO: set TX mode....
”));
    // record the current state.
    VAR(state) = 1;
}

/* This command sets the RFM component (radio) into receiving mode */
char TOS_COMMAND(RFM_RX_MODE)(){
    if(VAR(state) == 2) return 0;
    // set the RFM to RX mode.
    SET_RFM_CTL0_PIN();
    SET_RFM_CTL1_PIN();
    CLR_RFM_TXD_PIN();
    dbg(DBG_RADIO, (“RADIO: set RX mode....”));
    // record the current state.
    VAR(state) = 0;
}

}
Example code piece:

```c
/* This command sets timer1 to different sampling level */
char TOS_COMMAND(RFM_SET_BIT_RATE)(char level)
{
    if(level == 0)
    {
        #ifdef DOT
        VAR(precision) = 0;
        #endif
        outp(0x00, OCR1AH); // set upper byte of comp reg.
        outp(0xc8, OCR1AL); // set the lower byte compare
        outp(0x00, TCNT1H); // clear current counter value
        outp(0x00, TCNT1L); // clear current counter high byte value
    }
    else if(level == 1)
    {
        #ifdef DOT
        VAR(precision) = 0;
        #endif
        outp(0x01, OCR1AH); // set upper byte of comp reg.
        outp(0x2c, OCR1AL); // set the lower byte compare
    }
    else if(level == 2)
    {
        #ifdef DOT
        VAR(precision) = 1;
        outp(bit_timer >> 8, OCR1AH); // set upper byte of comp reg.
        outp(bit_timer & 0xff, OCR1AL); // set the lower byte compare
        #else
        outp(0x01, OCR1AH); // set upper byte of comp reg.
        outp(0x90, OCR1AL); // set the lower byte compare
        #endif
    }
    return 1;
}
```

**Summary**

Radio Handling at the bit level
Most of the bit level handling is designed around handling settings.
Bit level is not responsible for error/system checking for the most part.
Abstraction comes into play at a very high level.