Lesson 5: Host to Mote Communication

Mote Boot Camp

10/17/2001

Lesson 5: Host—Mote communication

- Motes communication over UART to PC
- Serial port must be configured to use 19200bps with NO FLOW CONTROL
- tools/listen.java provides as the simplest way to display data coming from the motes

Step 1:

- Lets confirm that your tools are installed correctly.
- Compile and install the oscilloscope application onto a mote
- Confirm that the yellow led is periodically blinking
- Plug the mote into a programming board and connect the programming board to your serial port

Step 2:

- Try to read from your serial port

```
cd tools
make
java listen COM1
```

- Expected output:

```
listen started printing all ports...
- COM3
- COM1
- LPT1
- LPT2
done.
baud rate: 9600
data bits: 8
stop bits: 1
parity: 0
baud rate: 19200
data bits: 8 stop bits: 1
parity: 0
7E 00 0A 7D 01 00 72 EE 01 00 5D 03 5A 03 5A 03 59 03 53 03 4B 03 4B 03 58 03 61 03 61 03 00 00 00 00 76 9B
7E 00 0A 7D 01 00 7C EE 01 00 61 03 66 03 67 03 69 03 67 03 63 03 64 03 64 03 64 03 65 03 00 00 00 00 84 E0
7E 00 0A 7D 01 00 86 EE 01 00 65 03 66 03 65 03 65 03 64 03 65 03 66 03 65 03 66 03 66 03 00 00 00 00 10 97
7E 00 0A 7D 01 00 90 EE 01 00 66 03 66 03 66 03 66 03 66 03 66 03 66 03 66 03 66 03 66 03 00 00 00 00 CD 5E . .
```

What are you seeing…

- OSCOPE application periodically:
  - Samples ADC
  - Collects multiple readings into a single packet
  - Sends the data collected over the UART
    - Double-buffered data collection is used to pipeline collection and transmission.

How do you interpret the packet?

- TOS Packet Structure:

```
struct TOS_Msg{
     unsigned int destionation_id;
     unsigned char handler_type;
     unsigned char group_id;
     unsigned char data[30];
     unsigned int crc;
 };
```

- Packet is 36 bytes long
  - 4 bytes header, 30 bytes data, 2 bytes CRC

```
7E 00 0A 7D 01 00 72 EE 01 00 5D 03 5A 03 5A 03 59 03 53 03 4B 03 4B 03 58 03 61 03 61 03 00 00 00 00 76 9B
```

How do you interpret the packet? (cont.)

- Data Payload Structure:

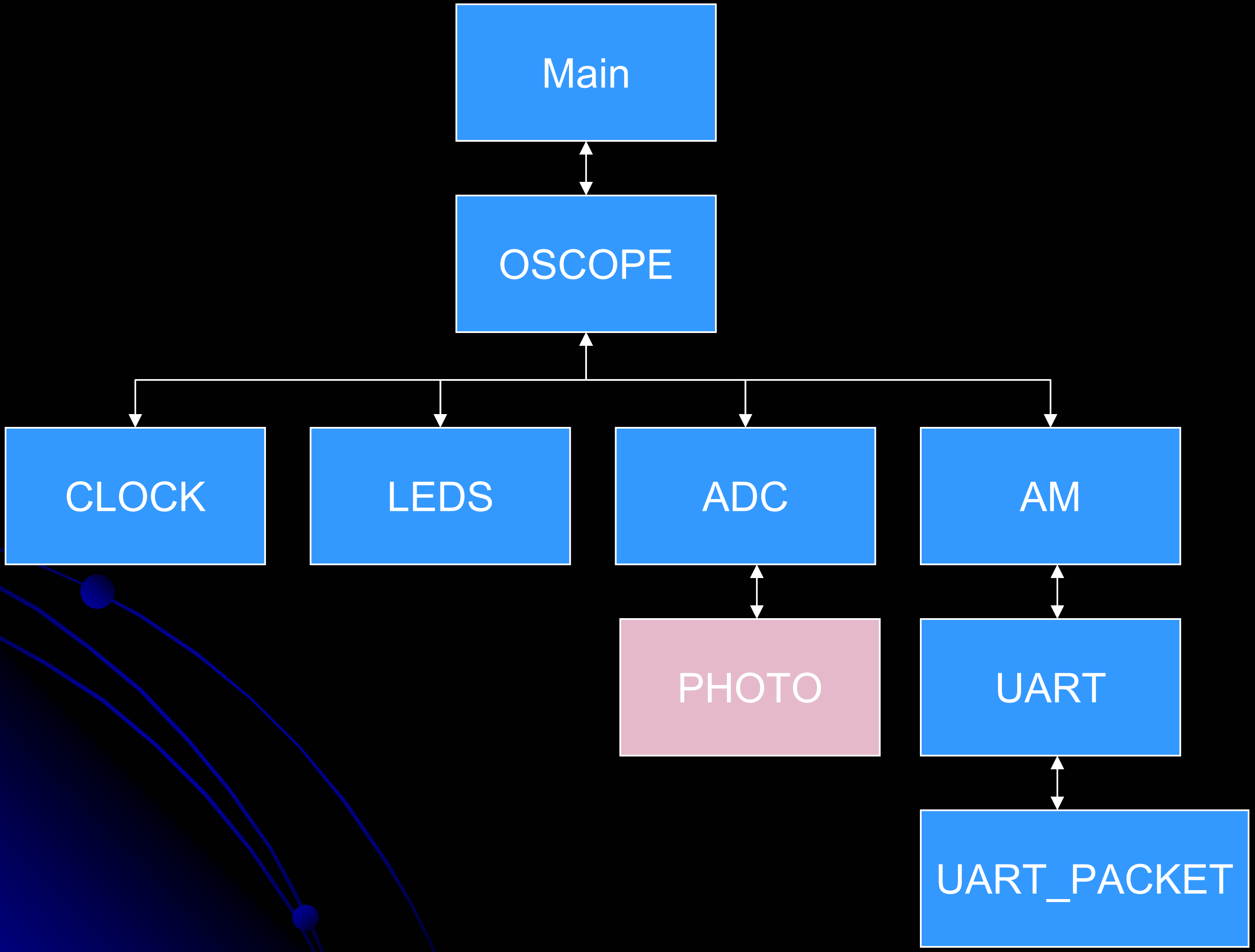
```
struct data_packet{
        unsigned int source_mote_id;
        unsigned int last_reading_number;
        unsigned int channel;
        int data[READINGS_PER_PACKET];
 };
```

- Data Payload Breakdown
  - 2 bytes source ID, 2 bytes reading_number, 2 byte channel, 2 byte pairs of readings

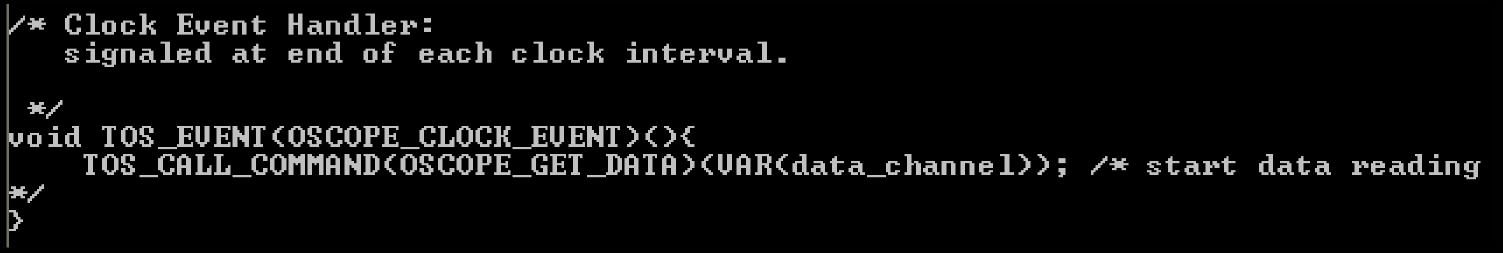
```
01 00 72 EE 01 00 5D 03 5A 03 5A 03 59 03 53 03 4B 03 4B 03 58 03 61 03 61 03 00 00 00 00
```

For ints, LSB first: 5D 03 = 0x035D = 861

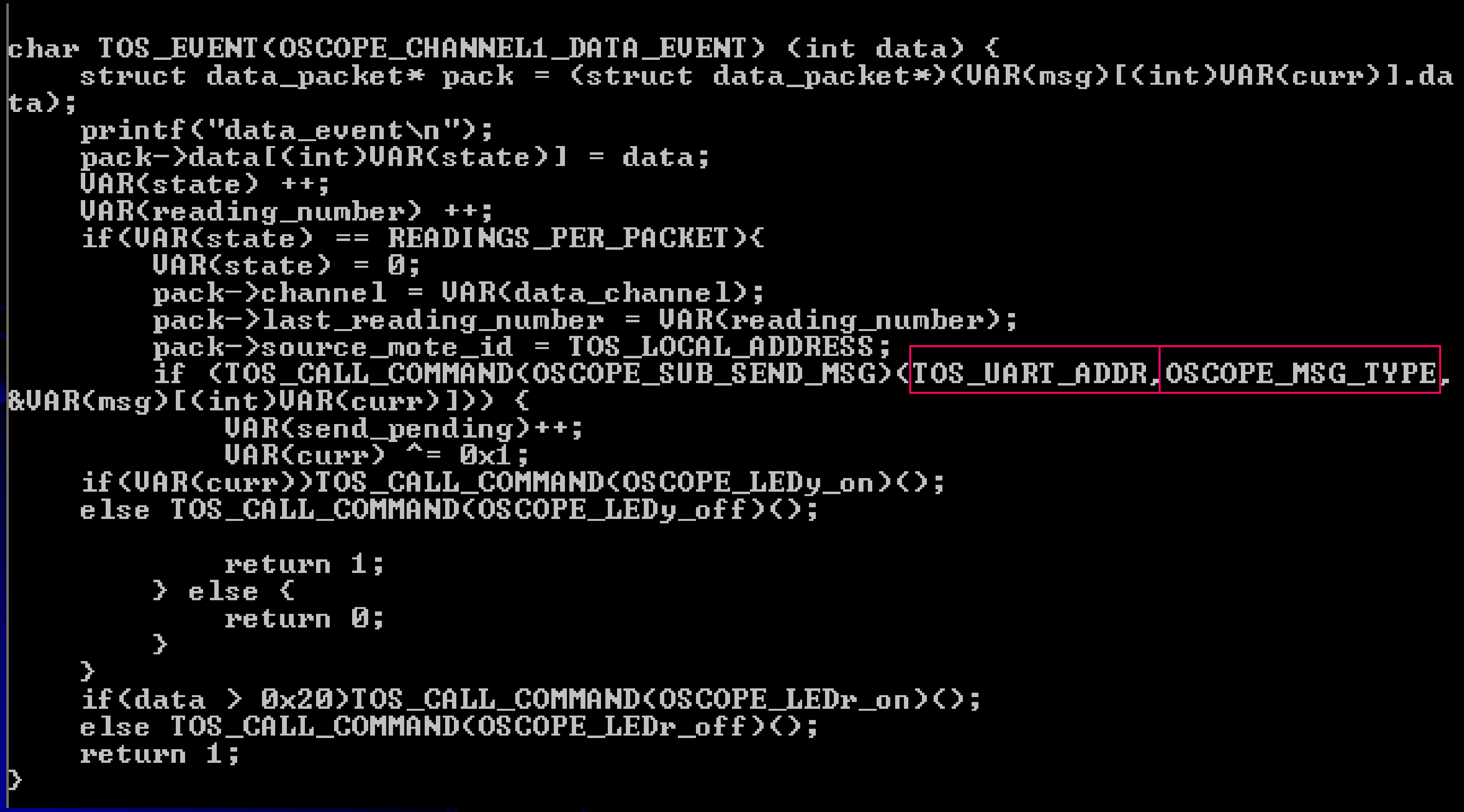
OSCOPE application graph



Key application code:

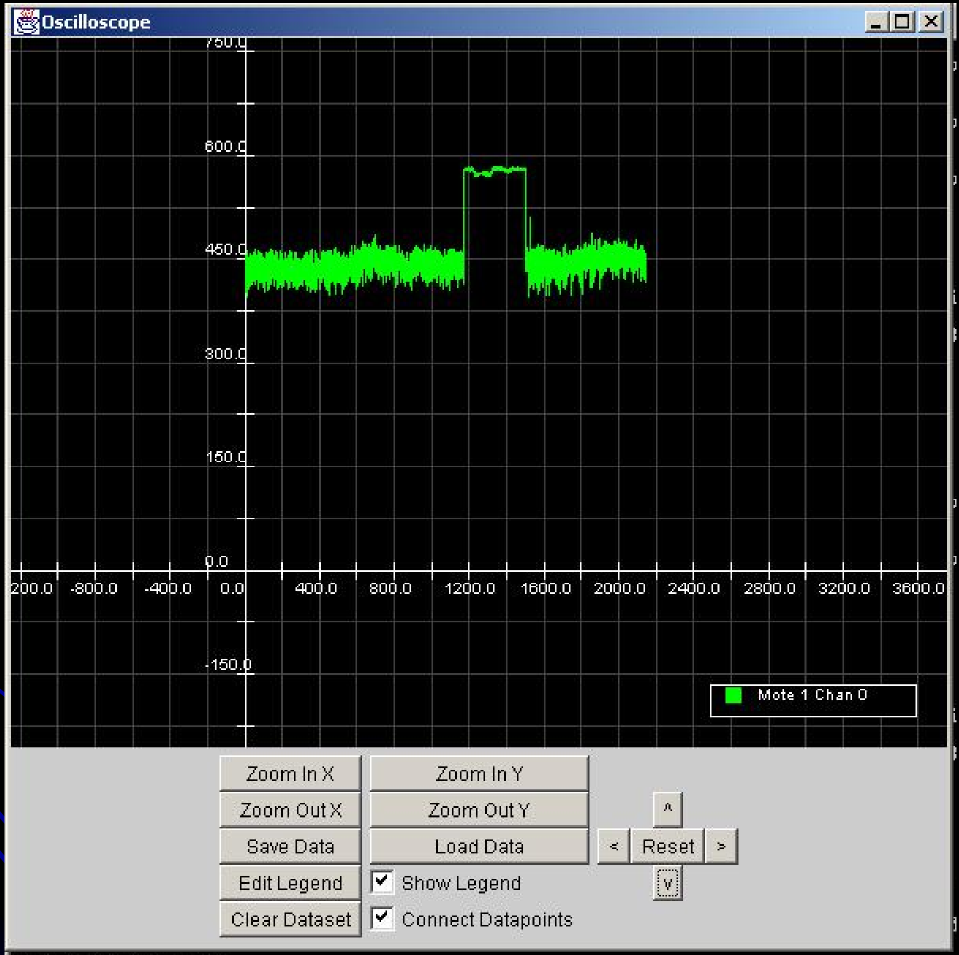


```
/* Clock Event Handler:
   signaled at end of each clock interval.

 */
void TOS_EVENT(OSCOPE_CLOCK_EVENT)(){
    TOS_CALL_COMMAND(OSCOPE_GET_DATA)(VAR(data_channel)); /* start data reading
*/
}
```



```
char TOS_EVENT(OSCOPE_CHANNEL1_DATA_EVENT) (int data) {
    struct data_packet* pack = (struct data_packet*)(VAR(msg)[(int)VAR(curr)].da
ta);
    printf("data_event\n");
    pack->data[(int)VAR(state)] = data;
    VAR(state) ++;
    VAR(reading_number) ++;
    if(VAR(state) == READINGS_PER_PACKET){
        VAR(state) = 0;
        pack->channel = VAR(data_channel);
        pack->last_reading_number = VAR(reading_number);
        pack->source_mote_id = TOS_LOCAL_ADDRESS;
        if (TOS_CALL_COMMAND(OSCOPE_SUB_SEND_MSG)(TOS_UART_ADDR,OSCOPE_MSG_TYPE,
&VAR(msg)[(int)VAR(curr)])) {
            VAR(send_pending)++;
            VAR(curr) ^= 0x1;
    if(VAR(curr))TOS_CALL_COMMAND(OSCOPE_LEDy_on)();
    else TOS_CALL_COMMAND(OSCOPE_LEDy_off)();

            return 1;
        } else {
            return 0;
        }
    }
    if(data > 0x20)TOS_CALL_COMMAND(OSCOPE_LEDr_on)();
    else TOS_CALL_COMMAND(OSCOPE_LEDr_off)();
    return 1;
}
```

An easier way to visualize data:



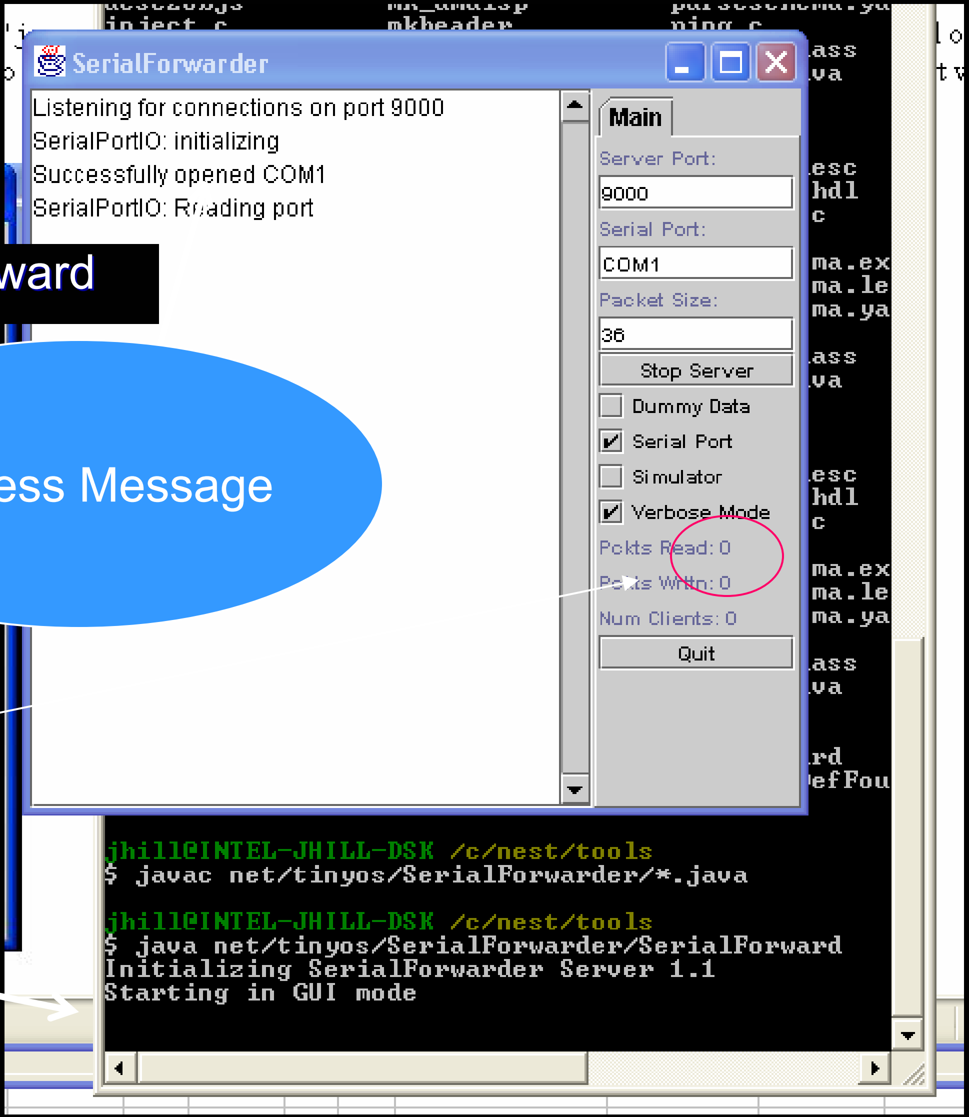
☑ Show Legend

☑ Connect Datapoints

Step 1: The Serial Forwarder

- Fisrt, start the SerialForwarder:

```
javac net/tinyos/SerialForwarder/*.java
java net/tinyos/SerialForwarder/SerialForward
```



```
Listening for connections on port 9000
SerialPortIO: initializing
Successfully opened COM1
SerialPortIO: Reading port
```

Main

Server Port: 9000

Serial Port: COM1

Packet Size: 36

☐ Dummy Data

☑ Serial Port

☐ Simulator

☑ Verbose Mode

Pckts Read: 0

Pckts Wrttn: 0

Num Clients: 0

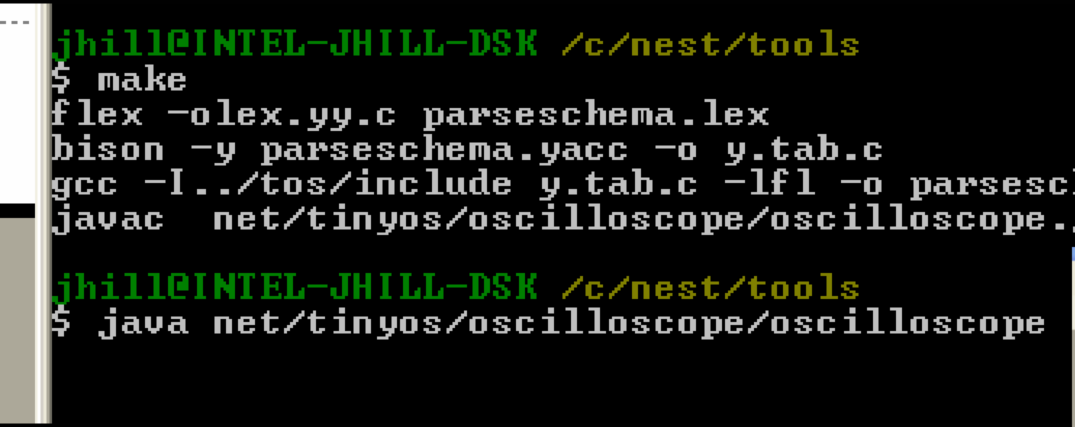
```
jhill@INTEL-JHILL-DSK /c/nest/tools
$ javac net/tinyos/SerialForwarder/*.java

jhill@INTEL-JHILL-DSK /c/nest/tools
$ java net/tinyos/SerialForwarder/SerialForward
Initializing SerialForwarder Server 1.1
Starting in GUI mode
```

Success Message

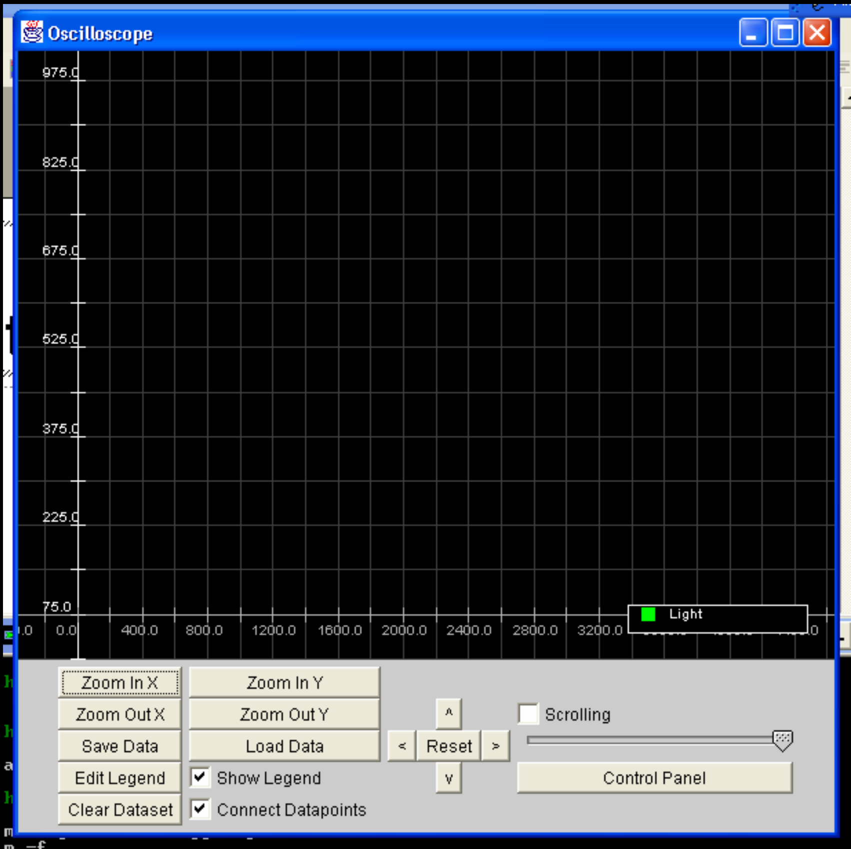
Pckts Read: should increment

No Error Messages

Step 2: The Oscilloscope Application



```
jhill@INTEL-JHILL-DSK /c/nest/tools
$ make
flex -olex.yy.c parseschema.lex
bison -y parseschema.yacc -o y.tab.c
gcc -I../tos/include y.tab.c -lfl -o parseschema
javac  net/tinyos/oscilloscope/oscilloscope.java

jhill@INTEL-JHILL-DSK /c/nest/tools
$ java net/tinyos/oscilloscope/oscilloscope
```



- Run the app from the tools directory

Generic Base

- Universal base station for communicating with motes
- app/generic_base acts as a bridge between radio and UART
  - All packets received on radio are forwarded to the UART
  - All pcakets received on UART are forwarded to the radio
  - GroupID checking is performed