

Bluetooth Control Interface for the PicoRadio Network

Muhua Pang

University of Rochester

**SUPURB SUMMER UNDERGRADUATE PROGRAM IN
ENGINEERING RESEARCH AT BERKELEY 2000**

Faculty Mentor: **Jan Rabaey**

Graduate Mentor: **Fred Burghardt**

Electrical Engineering and Computer Sciences

College of Engineering

University of California at Berkeley



ABSTRACT

The PicoRadio group at the Berkeley Wireless Research Center is conducting research into small, very low power sensor nodes that will operate in a wireless network environment. This PicoRadio Network will utilize various radio technologies. In order to adopt a new radio technology, a control interface is required. This report describes the process to design a control interface for the Bluetooth radio. The BlueTooth Control Interface integrates the low-cost and power efficient Bluetooth radio into the PicoRadio Test Bed environment. It allows a user to control a BlueTooth radio from a console menu and also allows the radio to be used as the communication link in an embedded application. The design is accomplished based on a preexisting interface to a Proxim frequency-hop radio, and it performs as expected under simulation.

1 Introduction

The goal of the PicoRadio group at the Berkeley Wireless Research Center is to implement a tiny, ultra low power, configurable system on a chip (SOC) that can be used in sensor networking, personal communications, smart environments, and other possible applications. In order to fit into other electronic devices easily, the size of the chip, sensor, and power source is desired to be smaller than 1mm^3 , cost less than \$1, and consume power less than $60\mu\text{W}$ [evolved from 1]. However, to proceed with system level research before the chip is available, one focus of the PicoRadio project is to build a prototype in a convenient size with flexible radio technology.

The basic structure of the PicoNode prototype consists of a battery power supply, an ARM microprocessor, a Xilinx FPGA (Field Programmable Gate Array: a set of memory and logic elements that can be programmed for a particular function by the user instead of manufacturer of the device [2]). The digital processing board contains the ARM microprocessor and the Xilinx FPGA. The power supply board provides voltage to both the radio and the digital board.

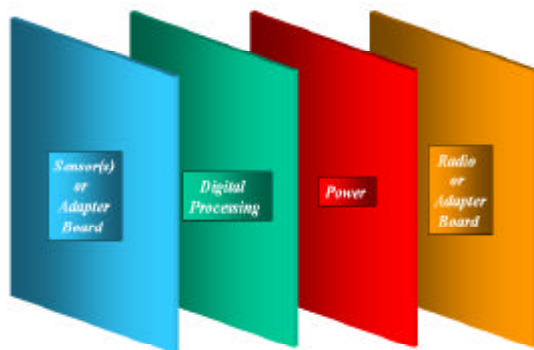


Figure1: Structure of a PicoNode
starting point.

To control the radio, a serial interface between the digital processing board and radio is required. The control interface is implemented as digital circuitry within the Xilinx, and its main function is to program control registers within the radio that determine transmit/receive frequencies, among other things. A previously designed interface for a Proxim radio is used as a

This paper presents the process undertaken to design and implement a control interface for the Bluetooth Radio. To accomplish the design and implementation of the Bluetooth radio control interface required clear understanding of the Proxim radio control interface.

2 Proxim Radio Control Interface

The Proxim Control Interface was composed of two major digital circuit blocks—a state machine and three shift registers, plus two minor circuit blocks—a phase clock and a ROM which stored the transmitting/receiving frequency values.

2.1 State Machine

The state machine precisely described the behavior of the digital circuits that controlled the radio in the form of a flowchart, and was the essential part of the control interface because it manipulated the registers in the radio.

There were eight registers in the Proxim radio: 4 normal registers (those normally programmed by users) and 4 hidden registers (those not normally programmed by users) [3]. Only two of the normal registers--Tune register and Preamble register were programmed in the design. The main function of the Tune register was to select transmitting/receiving frequencies for the radio. Binary value 000 specified the address of this register, and the 9-bit data values stored in the register represented different frequencies. The Preamble register, also known as the Toggle register, was addressed by the 3-bit code 011 and contained the 1-bit value 0 that indicated preable insertion enable; its function was to enable the transmitting mode.

The state machine first loaded and shifted the Tune register address and data serially to the radio controller to select a frequency. If transmitting was required, then the Preamble register address and data would be loaded. However, if receiving was required, it was not necessary to load the Preamble register. Note: SYNCLK, SYNLD, and SYNDATA were the three input signals that were generated by the state machine to the Proxim radio controller.

2.2 Right shift registers

The function of the shift registers was to load the Tune and Preamble registers' addresses and values and to serially shift them into the radio controller. The behavior of these registers was controlled by the state machine in terms of when to load and when to shift. Since the address and value of the Preamble register bits were always 0110, they were hard-wired to VCC and GND. The 12-bit frequency values of the Tune register were stored in ROM. Two 8-bit right shift registers and one 4-bit right shift register were connected serially to transport these values to the radio controller.

2.3 ROM

A 4x8 ROM was built to store the Tune registers frequency values and address, which could be accessed through the ROM address. Two of the frequency values were used for receiving, and the other two were used for transmitting. The ROM was implemented in VHDL code.

2.4 Phase clock

The phase clock generated the SYNCLK and SYNLD signals. The 20MHz system clock was divided by 16 to generate SYNCLK which was close to the radio's data rate of 1.6Mbps. Tune and Preamble register addresses and values were serially shifted to the radio on a positive SYNCLK edge one bit at a time. SYNLD was asserted when the shifting sequence was complete for each register at which time the radio would store the values. The exit condition from one state to another was also controlled by the phase clock.

Careful analysis of these circuit blocks in the Proxim radio control interface gave a very clear and specific description of what a control interface was and how it worked. Also the techniques used in the design would be very valuable for the Bluetooth interface design. Now, it's time to go to the next stage—Bluetooth Control Interface design.

3 Bluetooth Control Interface

3.1 Bluetooth radio technology

Bluetooth is a low-cost and power efficient radio chip that allows the elimination of cables between electronic devices by replacing them with wireless communication. It operates in the license-free Industrial Scientific Medical (ISM) band range from 2.4GHz to 2.5GHz [4]. To avoid interference from other electronic devices, Bluetooth technology adopted the frequency-hopping scheme where the radio transceiver hops from one channel to another in a pseudo random order. Fast hopping results in wide bandwidth usage with good immunity to interference. There are 79 hopping frequencies available with a TX/RX bit rate of 1 Mbits/s [4].

3.2 Design considerations

The Bluetooth serial control interface design is based on a preliminary report from Ericsson Corp, the designers of Bluetooth. There are four interconnections between the Bluetooth radio and Xilinx controller for register loading: control data input (SI_CDI), control mode select (SI_CMS), control clock (SI_CLK), and control data output (SI_CDO). SI_CDI transports data from the Xilinx to the radio while SI_CDO transports data back to the Xilinx; SI_CLK and SI_CMS control the transfer of data to and from registers in the radio. The register load state depends on the values of SI_CMS at the positive edge of SI_CLK (see figure 2).

3.3 Design Procedure

Nine registers exist in the radio controller; four are programmed in this design. The others use a default power-up value. At power-up, the programming sequence is:

1. Control register: allows Bluetooth control from the FPGA
2. CHP control register: enables a charge pump power supply
3. Enable register: enables various control functions
4. Channel register: contains a frequency value

Each of these registers has a 6-bit address and contains 8-bits of data. The following diagram illustrates how IR Scan and DR Scan are performed. Instruction Register (IR) scan selects a specific register to write to by scanning in the register address. Data Register (DR) scan causes a new value to be shifted into the register selected by the IR scan.

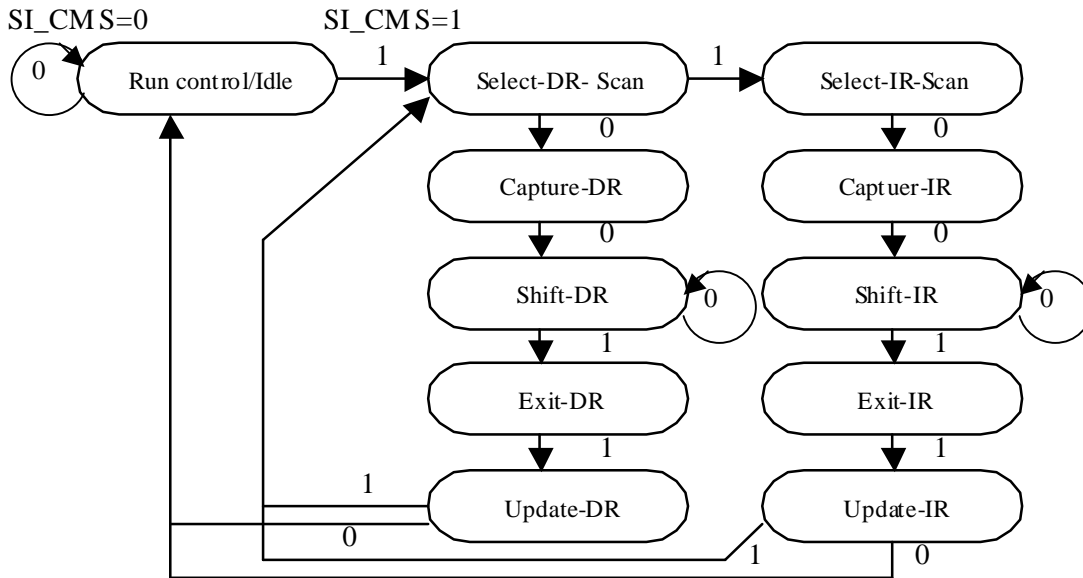


Figure 2: Flow chart for serial interface control

Using the flow chart in figure 2, the timing relationship between SI_CMS and SI_CDI can be determined. The transition from one state to another is directed by the values at SI_CMS when SI_CLK occurs. In the Shift_IR scan state, the address of the target register is shifted in (LSB first) on SI_CDI. Then in the Shift_DR state, the value of the target register is shifted in (LSB first) also on SI_CDI. Therefore, there are two data streams going into the radio, both 24 bits long. Two 24 bits serial shift registers (interface registers) can accomplish the task to shift them in.

As mentioned before, the Bluetooth register programming sequence at power-up is Control, CHP, Enable, and Channel. In order to accomplish this, a Finite State Machine (FSM) was designed. The function of the state machine is to load the address and data for each

Bluetooth register into the interface registers, then shift them into the radio according to this sequence.

The address and value of each Bluetooth register are loaded into the interface registers in one SI_CLK cycle and are shifted out in the next 23 clock cycles. A phase clock adapted from the Proxim interface is used to control the exit condition from the shifting state. After the Bluetooth Channel register address is programmed the first time, initialization is complete. To change frequency, it's not necessary to do another IR scan since the Channel register address is still in IR; only a new DR-scan is needed. Therefore, only the last thirteen bits need to be shifted to the radio controller to change frequency (see 3.4.2 for details). The state machine normally sits in the Idle state if no new frequency is requested. For frequency hopping, when the HOP signal is high, a new frequency value is read from a ROM and programmed into the radio. The ISM standard requires at least 75 hopping frequencies are used. When the address of the ROM which contains the frequency values reaches 75, it wraps back to address 0 and starts the hopping sequence again. At power up or reset, the state machine behavior should be as follows:

1. Reset state
2. Load in the address and data of the Control register
3. Shift the address and data of the Control register to the radio controller
4. Load in the address and data of the CHP Control register
5. Shift the address and data of the CHP Control register to the radio controller
6. Load in the address and data of the Enable register
7. Shift the address and data of the Enable register to the radio controller
8. Load in the address and data of the Channel register
9. Shift the address and data of the Channel register to the radio controller
10. Loop in Idle state

11. When a new frequency is requested, increment the address of the ROM by 1, and load in the frequency value
12. Shift the frequency value to the radio controller
13. Return to Idle state

For a detailed state machine diagram, refer to figure 3.

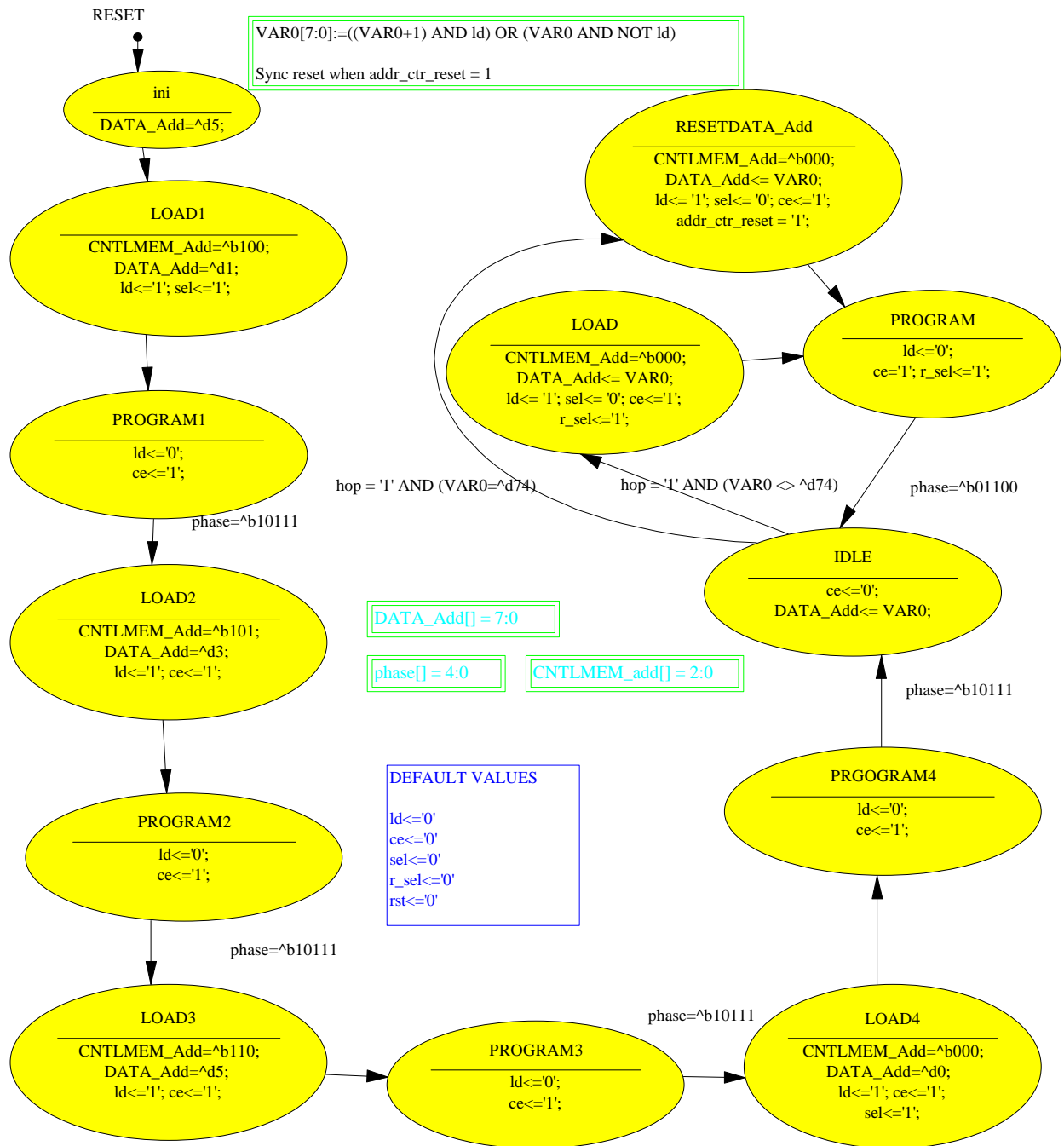


Figure 3: State Machine for Bluetooth Control Interface

Like in the Proxim interface, the phase clock provides timing for the circuit. For instance, the phase clock generates SI_CLK and controls the exit condition for the state machine. The designer arbitrarily chose to generate SI_CLK on phase 3 (1/16 system clock) because this is what Proxim interface used. Testing needs to be done to refine the choice of clock speed. The initial phase counter is set to 00000 before any program state is entered. When it turns to 10111, the state machine exits from the PROGRAM state for the first four loads. However, for frequency hopping, since the exit happens after 12 clock cycles, the exit condition for the phase counter is set to 01100. Note: The phase counter has to be set back to 00000 in every load state to make the problem simpler, otherwise the exit condition would be different for each PROGRAM state. (see figure 3)

3.4 Implementation Process

One goal of this project is to learn the difference between design and implementation. So how are these two processes different? In general, the design process is where the designer defines and verifies the system behavior while the implementation process is where the designer chooses the target technology, and tools such as VHDL code or schematics, and creates the system itself. The following subsections describe the implementation methods used to accomplish the above design ideas.

3.4.1 State Machine

The state machine is implemented with the aid of a software package called Viewlogic. The software allows a state machine to be described graphically, and then automatically generates VHDL code and a schematic symbol based on the state diagram. Therefore it's critical to draw the state diagram right. One way to check the behavior of the state machine is to run a simulation.

3.4.2 Shift Registers

The shift registers are implemented as schematic blocks. For an IR scan followed immediately by a DR scan, the SI_CMS bit stream repeats every 24 clock cycles, therefore, the bits are hard-wired to VCC and GND. According to Figure 2, SI_CMS controls the state transition in the order 11000000011100000000110; correspondingly, SI_CDI bits are xxxxaaaaaaxxxxvVVVVVVVvxx (x=don't care, a=register address, v=register value). All the x values are hard-wired to GND since they are not involved in the design. For a DR scan only, the last thirteen bits of SI_CMS bit stream need to shift out serially and they repeat every 13 clock cycles in the order 100000000110; correspondingly, SI_CDI bits are xxxvVVVVVVVvxx.

For the first four loads, an IR scan is followed by a DR scan. Afterward, in the frequency hopping loop, only a DR scan is necessary (refer to figure 3). A 2x1 Multiplexier extracted from Xilinx library is implemented to control this situation. Two 24-bit serial left shift registers need to be implemented for SI_CMS and SI_CDI, but they are not available in the FPGA library, so 6 4-bit left shift registers are used instead. Load, Clock Enable, and Clock signals are tied together for all the registers because SI_CMS and SI_CDI load and shift at the same time.

3.4.3 Phase Clock

Since the state machine operates according to SI_CLK, phase 3 should be the clock to the phase counter which generates the phase exit condition. To implement the phase counter, a 4 bit counter for the SI_CLK and an 8 bit counter for exit condition are used. The symbols for these counters are available from the Xilinx library.

3.4.4 ROMs

There are two ROMs created for the design. One holds the addresses and default data of all the radio registers while the other one holds the frequency values. Seven writable registers exist in the radio; the address of each register is 6 bits long and the initial value is 8 bits long. Therefore, One 8x12 ROM (8 is the size of the address and 12 is the size of data outputs) is

implemented to store the address and initial value of all the registers. One 128 \times 8 ROM contains the Channel register values for a 75 frequency hop sequence.

Both ROMs are implemented in VHDL code. VHDL is a hardware language used to describe the behavior of digital circuitry. Again with the aid of Viewlogic, a schematic symbol is generated automatically to represent the VHDL behavior so that it can be embedded in a schematic. One thing to keep in mind is that according to the state machine the first four loads are from the 8 \times 12 ROM, afterward the loads are from the frequency value ROM. Therefore, a 2 \times 1 Multiplexer is used to make the selection.

After completion of each of these circuit blocks, they are put into one schematic to simulate. The simulation waveform gives a very clear description of how the circuits behave in term of timing. Then corrections are made to meet the expectations. For example, at first the shift registers were designed to shift in the right direction, in which case the designer didn't realize that the register values got shifted in first instead of the address, of course the radio controller wouldn't know where to put these values unless the address was specified first. To solve the problem, the shifting direction was changed to left. This is just one example of corrections that were made. The design is constantly revised until the expected simulation waveform is achieved.

4 Conclusion

The design and implementation of a control interface for Bluetooth Radio is successfully complete. Simulation runs as expected. Within the near future, bench testing shall be conducted to verify the design in a real world time frame. The design will be used as an integral part of the PicoRadio Test Bed for the remainder of the program.

5 Acknowledgements

I would like to thank Professor Jan Rabaey for giving me the opportunity to work with

the PicoRadio group. A special thanks goes to Fred Burghardt whom I am very honored to have as a mentor. With all his support and guidance, I am able to go through the design process and to present the ideas here. Lastly, I want to thank Dr. Sheila Humpreys, Marie Mayne, Monica Lin, Susanne Kauer, and all the other staff workers in the SUPURB program for making the program such an enjoyable experience.

6 References

- [1] J. Rabaey, M Ammer, J. L.da Silva Jr, D. Patel, and S. Roundy. "PicoRadio Supports Ad Hoc Ultra-Low Power Wireless Networking." IEEE Computer. Jul. 2000, pp.42-47.
- [2] J. M. Yarbrough. 1997. Digital Logic Applications And Design. St. Paul: West Publishing Company.
- [3] "The Berkeley Wireless Research Center PicoRadio Project."
<http://bwrc.eecs.berkeley.edu/Local/Proxim>.
- [4] J. Haartsen. "Bluetooth—The universal radio interface for *ad hoc*, wireless connectivity." No.3, Ericsson Review. 1998, pp.110-117.