FPGA implementation of a ZigBee wireless network control interface to transmit biomedical signals

M A Gómez López¹,²,³, C B Goy¹,²,³, P C Bolognini² and M C Herrera²,³

¹ Departamento de Electricidad, Electrónica y Computación, Facultad de Ciencias Exactas y Tecnología, Universidad Nacional de Tucumán, Av. Independencia 1900, Tucumán, Argentina
² Laboratorio de Investigaciones Cardiovasculares Multidisciplinario, Facultad de Ciencias Exactas y Tecnología, Universidad Nacional de Tucumán, Av. Independencia 1900, Tucumán, Argentina
³ Instituto Superior de Investigaciones Biológicas (INSIBIO), Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), Chacabuco 461, Tucumán, Argentina

E-mail: mgomezlopez@herrera.unt.edu.ar

Abstract. In recent years, cardiac hemodynamic monitors have incorporated new technologies based on wireless sensor networks which can implement different types of communication protocols. More precisely, a digital conductance catheter system recently developed adds a wireless ZigBee module (IEEE 802.15.4 standards) to transmit cardiac signals (ECG, intraventricular pressure and volume) which would allow the physicians to evaluate the patient’s cardiac status in a noninvasively way. The aim of this paper is to describe a control interface, implemented in a FPGA device, to manage a ZigBee wireless network. ZigBee technology is used due to its excellent performance including simplicity, low-power consumption, short-range transmission and low cost. FPGA internal memory stores 8-bit signals with which the control interface prepares the information packets. These data were send to the ZigBee END DEVICE module that receives and transmits wirelessly to the external COORDINATOR module. Using an USB port, the COORDINATOR sends the signals to a personal computer for displaying. Each functional block of control interface was assessed by means of temporal diagrams. Three biological signals, organized in packets and converted to RS232 serial protocol, were successfully transmitted and displayed in a PC screen. For this purpose, a custom-made graphical software was designed using LabView.

1. Introduction
In recent years, the increasing development of wireless communication protocols has allowed the inclusion of these technologies to all types of medical devices [1]-[4]. The information, obtained from various sensors or systems located on/inside patients, is transmitted through communication networks to remote stations where a medical specialist or an intelligent system takes or process the data. From these data, the physician-patient interaction will be established, depending on the application, either

4 To whom any correspondence should be addressed.
Cardiac hemodynamic monitors are some examples of these technologies. Some cardiac variables - such as the right ventricular pressure, right ventricle pre-ejection time and systolic time interval - are sensed in/on the body of patients - heart, arteries, veins - and transmitted by wireless systems to a monitoring station [5]-[9].

Recently, our research group has developed a cardiac hemodynamic monitor based on conductance catheter system (CCS). More precisely, a digital conductance catheter system (DCCS), completely implemented in a FPGA (Field Programmable Gate Array), has been designed and tested experimentally [10]. It should be a platform to optimize the conductance catheter system in order to develop an implantable device [11]. This platform adds a wireless communication module with which the researcher can access the patient's hemodynamic status noninvasively.

In general, the technical features required of the wireless communication protocols used in medical equipment are the short range and low power consumption. ZigBee is a protocol that ensures these specifications under the IEEE 802.15.4 Standard [12].

With this background, this paper describes the FPGA-embedded logical design of a control interface wireless network that allows transmission of cardiac hemodynamic signals from the FPGA internal memory to a personal computer for visual presentation.

2. Design

Briefly, newly conductance-catheter equipment DCCS is a development system based on left intraventricular volume (LIVV) determination by recording multiple intracavitary conductances, left intraventricular pressure (LIVP) using a miniature ceramic transducer and a simultaneous electrocardiographic signal (ECG) [10]. All of them are digitized with 8 bits and sampled at 432Hz. Fully implemented in a FPGA, DCCS stores these signals in a 1005x8 RAM memory and adequates the data to the ZigBee protocol communication.

FPGA is an embedded programmable logic device FLEX10k70RC240 with SRAM (Static Read Access Memory) technology [Altera Corp, 2003. FLEX 10K. Embedded Programmable Logic Device Family] which is part of an educational board (UP2 Education Board, Altera). Inside FPGA, the logic equations are implemented using MAX+PLUS II development system. Schematically, the design is realized by means of Altera’s hardware language description (AHDL).

In turn, FPGA is connected to the wireless transmission kit (Kit XBee ZB, ZigBee-PRO) which has two modules, the first called END DEVICE and the second called COORDINATOR. The first module can operate in low power consumption mode and the second receives and sends the data, using an USB2UART port, to a PC where, finally, data are displayed on a screen. These modules are reconfigurable modems that implement standard ZigBee communication protocols. Figure 1 shows a whole schematic diagram of the system.

Furthermore, the ZigBee kit includes two USB2UART boards which, in turn, could have a serial port from an USB port. Moreover, both modules are operated in API mode (Application Programming Interface) that allows a better management of network properties.

![Figure 1. Block diagram of the proposed system. See text for explanations.](image-url)
3. API message structure

Figure 2 presents a complete message configuration necessary to transmit by the wireless system.

| Frame Fields           | Offset (Byte) | Example | Description                                                                 |
|------------------------|---------------|---------|-----------------------------------------------------------------------------|
| Start Delimiter        | 0             | 0x7E    | Number of bytes between the length and the checksum                         |
| Length                 | MSB 1         | 0x00    | Number of bytes between the length and the checksum                         |
|                        | LSB 2         | 0x30    |                                                                             |
| Frame Type             | 3             | 0x10    |                                                                             |
| Frame ID               | 4             | 0x01    |                                                                             |
| 64-bit Destination     | MSB 5         | 0x00    | Set to the 64-bit address of the destination device.                         |
| Address                | 6             | 0x13    |                                                                             |
|                        | 7             | 0xA2    |                                                                             |
|                        | 8             | 0x00    |                                                                             |
|                        | 9             | 0x40    |                                                                             |
|                        | 10            | 0x3A    |                                                                             |
|                        | 11            | 0x4A    |                                                                             |
|                        | 12            | 0x39    |                                                                             |
| 16-bit Destination     | MSB 13        | 0xFF    | Set to the 16-bit address of the destination device, if known. Set to 0xFFFFE if the address is unknown, or if sending a broadcast. |
| Network Address        | LSB 14        | 0xFE    |                                                                             |
| Broadcast Radius       | 15            | 0x00    |                                                                             |
| Options                | 16            | 0x00    |                                                                             |
| RF Data                | 17            | ...     | Data that is sent to the destination device                                  |
|                        | ...           | ...     |                                                                             |
|                        | 48            | ...     |                                                                             |
| Checksum               | 49            | ...     | 0xFF - the 8 bit sum of bytes from offset 3 to this byte.                    |

Figure 2. Message configuration including API-specific structure. Detail Data Structure (in API mode). Reproduced from IEEE 802.15.4 Standard [12].

Each API packet consists of a header with 17 memory address or bytes -from position 0 to 16- (see figure 2) and a set of data –RF Data– (LIVP, LIVV and ECG signals) that are located from position 17 to 48 (33 bytes) and, finally, one byte for the Checksum (see details later in the text).

4. System description

Figure 3 shows the logical diagram incorporated in the FPGA. Digital data -corresponding to biological signals- are stored in ROM memory (1 byte per data); in turn, this memory delivers the data to a read/write RAM memory where the API packet is built. The control needed to build each signal packet is implemented by the MESSAGE CONTROL block that creates both, CLK1 and CLK2, from the FPGA main CLOCK.

Also, COUNTER MOD 1001 provides the address lines to ROM memory while COUNTER MOD 56 provides 56 address lines to RAM memory, both activated by CLK1. These blocks allow to create the information packets with a specific format. Furthermore, ADDER block generates a control signal that defines the header and the position of the data and sets an interval between packets to avoid overlap. Finally, TRANSMITTER converts the data to RS232 serial protocol required to access to the END DEVICE module (see figure 1).

When the transmission is requested (START in HIGH), the bytes related to each API message are extracted from the RAM, one by one. Thus, these bytes are converted to RS232 asynchronous serial protocol and are buffered at the END DEVICE module until they are transmitted wirelessly to the destination address specified in the API message header. Besides, when START is in LOW; the END DEVICE module is “sleeping” (LOW state), so it has very low power consumption.
4.1. MESSAGE CONTROL
MESSAGE CONTROL is a logical block that sets a counter (28847 counts). It has two input signals; one is the 25.175MHz CLOCK (from the UP1 Education Board) and the other is the START that enables the transmission. The MESSAGE CONTROL outputs are two clocks of 872Hz (CLK1) and 9597Hz (CLK2), respectively, both implemented with lpm_Counter function (MAX+PLUS II, Altera). CLK1 sets the loading/downloading frequency of ROM and RAM memories, whereas that CLK2 sets transmission speed of UART port.

Figure 4 shows the MESSAGE CONTROL temporal response including CLK1 and CLK2 signals. If START is in HIGH, the counters do not count; if it is in LOW, they count normally. START has a major role as it is the signal that initiates the transmission and “freezes” the whole system when it is enabled. Note that every eleven pulses of CLK2, there is only one pulse of CLK1. During these eleven pulses, one data (one byte) is converted according to RS232 protocol.

4.2. COUNTER MOD 1001
Figure 5 presents the COUNTER MOD 1001 functional block; it provides the address lines of the ROM memory (10bits bus called C1001 [9..0]). ROM memory stores the 34 bytes API packet. Implemented with lpm_Counter function (MAX+PLUS II, Altera), COUNTER MOD 1001 counts from 0 to 1000 where START enables the count.

Besides, COUNTER MOD 1001 has two clocks. CLOCK1 is the counter “pacemaker” with a frequency of 872Hz. The first 17 accounts match with the 17 bytes of the HEADER and one additional pulse necessary for synchronization. From the 19 account, CLOCK1 counts 34 pulses for 34 bytes (data). The following 4 pulses are used to generate a minimum waiting time interval called GUARD.
that the ZigBee module specifies for a successful transmission. The other clock, CLOCK_EN, is
designed to generate the timing as described above; that is, CLOCK_EN disables CLOCK 1 account
both during the 17 pulses of HEADER and during the 4 pulses of GUARD.

Figure 5. COUNTER MOD 1001 inputs and outputs (function of time). A- 0 to 13 counts. B- 26 to
36 counts. This behavior is periodically repeated every 56 pulses of CLK1.

4.3. ROM 1001x8
Figure 6 shows the ROM 1001x8 functional block implemented with lpm_rom function (MAX+PLUS
II, Altera). The input is 10 bits-bus C1001 [9..0] or the ROM memory address. The output SIGNAL
[7..0] has 1001 bytes of ECG, LIVP and LIVV, in that order. It should be noted that during
the GUARD and HEADER intervals, logical block output retains the last biological signal value.

Figure 6. ROM 1001X8 Output vs Time. A- 0 to 14 address. B- 26 to 34 address.

4.4. COUNTER MOD 56
Figure 7 presents the COUNTER MOD 56 functional block implemented with lpm_Counter function
(MAX+PLUS II, Altera). Its inputs are START and CLK1 signals and its output is a 6 bits-bus address
line (ADDRESS [5..0]). COUNTER MOD 56 counts from 0 to 55 and provides 56 address lines to the
RAM memory. Again, START enables the count.

Figure 7. COUNTER MOD 56 temporal signals. It only displays 0 to 31 counts. Identical behavior is
observed for 32 to 55 counts.
4.5. RAM 56x8

Figure 8 describes the RAM 56 functional block implemented with lpm_ram_dq function (MAX+PLUS II, Altera). It prepares the data for the TRANSMITTER logical block that sends them to the ZigBee module. RAM 56’s inputs are SIGNAL [7..0], ADDRESS [5..0] and CHECKSUM [7..0] while RAM 56’s outputs are Q [7..0], TX and CLKE.

Q [7..0] has the contents of the addresses listed in ADDRESS [5..0]. If ADDRESS [5..0] is minor than 18, RAM 56 block reads the contents of a memory initialization file (.mif) that includes the 17 values of message-header (HEADER). However, if ADDRESS [5..0] varies between 18 and 51, RAM 56 block writes the input data from SIGNAL [7..0] in these addresses. Finally, if ADDRESS [5..0] is equal to 52, RAM 56 block writes in the 52 memory address the value present in CHECKSUM [7..0] which is obtained at the output ADDER block described below.

TX output is in HIGH level when the data are transmitted; otherwise, TX output is in LOW. Also, CLKE output is in LOW during the GUARD time interval; otherwise, CLKE output is in HIGH. In addition, during GUARD interval, 0x00 hexadecimal value is placed at Q [7..0] output.

TX transition –from HIGH to LOW level- allows to control the GUARD time interval necessary to avoid saturating the END DEVICE buffer. In turn, GUARD interval time is obtained putting the TRANSMITTER output in HIGH level when TX is in LOW level.

TX, CLKE and START signals are entered into a three-input AND gate and its output is CLK_EN, the COUNTER MOD 1001 enable line. If any of these input signals are in LOW, CLK_EN too; hence, COUNTER MOD 1001 holds the account.

4.6. ADDER

Figure 9 shows the ADDER functional block implemented with lpm_add_sub function (MAX+PLUS II, Altera). ADDER’s inputs are Q[7..0], START and CLK1 while ADDER’s outputs are the bus CHECKSUM [7..0] and CONTROL.

ADDER block main function is to implement the CHECKSUM [7 .. 0] byte (see byte 49, figure 2); it is calculated as:

\[
\text{CHECKSUM} [7..0] = \text{HEX } "FF" - \Sigma \text{ offset 3 to 48 bytes}
\]
\[ \Sigma \text{offset 3 to 48 bytes} = \text{adder}_{i+1}[7..0] = \text{adder}[7..0] + Q_i[7..0] \]  

where the subscript \( i \) indicates the current value.

\[ \text{adder}[7..0] \]

\[ \text{checksum}[7..0] \]

**Figure 9.** ADDER temporal signals. \( \text{mef} \), internal variable that identifies the finite state machine.

Implemented with a 56 finite state machine, \( \text{adder}[7..0] \) is an 8 bits internal variable accumulating the partial sums. START input is connected to the RESET of the finite state machine starting it. First, the cumulative sum of 3 to 48 bytes of data and then, the difference indicated in equation (1) are calculated. In both cases, the results are truncated keeping only the lowest 8 bits. In HEADER time interval (first 18 states), \( \text{adder}[7..0] \) is initialized with HEX “00”; therefore, applying equation (1), CHECKSUM \([7..0]=\text{HEX} “FF”. The same occur during GUARD interval (54 to 56 pulses). During 19 to 53 states, the cumulative sum \( \text{adder}[7..0] \) is calculated with the bytes that enter sequentially from \( Q[7..0] \) applying equation (2); CHECKSUM \([7..0] \) is determined using (1). Finally, CONTROL signal function is to send a HIGH level pulse when ADDER is in the first state; it serves to measure the time interval between packets (not included in figure 9).

4.7. TRANSMITTER

Figure 10 presents the TRANSMITTER functional block implemented with HDL (Finite State Machine, MAX+PLUS II, Altera). Its inputs are \( Q[7..0] \), START and CL while its output is the \( T_x \) bit.

\[ \text{clk2} \]

\[ \text{start} \]

\[ \text{tx} \]

\[ \text{r}\]

\[ \text{t}_x \]

**Figure 10.** TRANSMITTER temporal signals. \( \text{mef} \), internal variable that identifies each state of the finite state machine.
In TRANSMITTER, Q[7..0] bytes are converted to RS232 asynchronous serial protocol adopting a LOW level for the “Start Bit” and a HIGH level for the “Stop Bit” without parity bit. 

It is implemented with an 11 finite state machine synchronized with CLK2 (9597Hz). The sequence begins at Initial State (see figure 10). This finite state machine is activated if the TX signal (from the RAM 56x8 block, see item 4.5) is in HIGH level; otherwise, T_x output is in IDLE state (inactive). This control assures that the TRANSMITTER block is not sending information during the GUARD interval. Besides, this machine begins when the START signal is in LOW; this control assures that the TRANSMITTER block is synchronized with the global enable signal.

5. Results

5.1. About the FPGA Logic Resources

After the FPGA design was completed, the resource utilization indices were calculated and presented in Table 1. A low percentage of used FPGA resources is observed except for embedded memory blocks (EABs) that store the data. However, ROM memory contents are refreshed when 1001 words were transmitted, therefore, does not affect the availability for future improvements. The few resources used by the ZigBee interface allows successfully incorporate the design to the DCCS.

| Parameter            | Used Resources |
|----------------------|----------------|
| Total dedicated input pins used | 1/6 (16%)     |
| Total I/O pins used  | 3/183 (2%)     |
| Total logic cells used | 241/3744 (6%) |
| Total embedded cells used | 56/72 (77%)   |
| Total EABs used      | 7/9 (77%)      |
| Total input pins required | 2             |
| Total output pins required | 2              |
| Total flipflops required | 121            |
| Synthesized logic cells | 55/3744 (1%)  |
| Memory bits          | 9312 (50%)     |
| Registered performance | 13.33MHz      |

5.2. About the transmitted data

Figure 11a shows the oscilloscope screen (software supplied by Tektronix). Clearly, it shows one packet of data (Channel 1) for each positive CONTROL pulse (Channel 2). The Channel 1 signal is the COORDINATOR received messages or PC input as a function of time.

To display the transmitted information, a graphical interface was designed using LabView software and VISA library (Configure Serial Port). It is able to extract only data of interest for each received message rejecting the headers and transmission technical information. In this particular design, 50 bytes messages are received via the RS232 port, where the relevant data are located from the fifteenth byte (Figure 11b). Note that the transmitted packets size (END DEVICE) differs from the received packets size (COORDINATOR) depending of the ZigBee protocol.

Figure 11b presents three transmitted biological signals from the FPGA-embeded DCCS, through the ZigBee modules, to the personal computer. Thus, the ECG, LIVP and arterial pressure were successfully displayed on a personal computer screen. These signals were compared with the signals in the ROM memory; missing data were not detected (loss of information) neither distortion of them.
Figure 11a. TEKTRONIX oscilloscope screen (viewed on the computer). Channel 1: COORDINATOR received messages or PC input as a function of time. Channel 2: FPGA CONTROL output.

Figure 11b. Wirelessly transmitted biological signals (from the FPGA to the PC via ZigBee) visualized by custom-made software (developed in LABVIEW). LIVP, ECG and Arterial Pressure are presented (superimposed).

6. Conclusion
A control interface for a ZigBee wireless network using FPGA logical design has successfully implemented. This interface receives 3 simultaneous cardiac signals from a digital conductance catheter system previously developed, generates data packets with a specific format and sends them to the transmitter device (END DEVICE) of a ZigBee module. This device wirelessly transmits the data to a COORDINATOR module. The latter is connected to a personal computer where data are displayed. Results demonstrate that the control interface performance is correct as it successfully recovers the transmitted signals.

Acknowledgments
This work was supported in part by grants from the UNT Research Council (CIUNT), 26/E422-3 Research Program and INSIBIO (CONICET).

References
[1] Cai K and Liang X 2010 A Zigbee based mesh network for ECG monitoring system 4th Int.Conf. on Bioinformatics and Biomedical Engineering (iCBBE) (Guangzhou: China Inf. Coll. Zhongkai Univ. of Agric. & Eng) 1-4
[2] Jung J and Lee J 2008 Device access control and reliable data transmission in zigbee based health monitoring system IEEE 10th Int. Conf. on: Advanced Communication Technology (Seoul: Electron. & Telecommun. Res. Inst.) 795-97
[3] Junnila S, Defee I, Zakrzewski M, Vainio A and Vanhala J 2008 UUTE home network for wireless health monitoring IEEE Int. Conf. on: Biocomputation, Bioinformatics, and Biomedical (Tampere: Dept. of Signal Process. Tampere Univ. of Technol.)125-30
[4] Kuryloski P, Giani A, Giannantonio R, Gilani K, Gravina R, Seppa V, Seto E, Shia V, Wang C, Yan P, Yang A, Hyttinen J, Sastry S, Wicker S and Bajcsy R 2009 Dexternet: an open platform for heterogeneous body sensor networks and its applications 6th IEEE Int. Workshop on Wearable and Implantable Body Sensor NetwK (New York: Ithaca Dept. of ECE Cornell Univ.) 92-7
[5] Bennett T, Kjellstrom B, Taepeke R and Ryden L 2005 Development of implantable devices for continuous ambulatory monitoring of central hemodynamic values Heart Failure Patients Rev. 28 573–84
[6] Steinhaus D, Reynolds D, Gadler F, Kay G, Hess M and Bennett T 2005 Implant experience with an implantable hemodynamic monitor for the management Symptomatic Heart Failure 28 747-53

[7] Yu C, Wang L, Chau E, Hon-Wah Chan R, Kong S, Tang M, Christensen J, Stadler R and Lau C 2005 Intrathoracic impedance monitoring in patients with heart failure. Correlation with fluid status and feasibility of early warning Preceding Hospitalization Circulation 112 841-48

[8] Tallaj J, Singla I and Bourge R 2011 Implantable hemodynamic monitors Cardiology Clinics 29 (2) 289-99

[9] Walton A and Krum H 2005 The heartpod implantable heart failure therapy system Heart Lung and Circulation 14S S31–3

[10] Gómez López M 2008 Equipo para mediciones en el sistema cardiovascular: Tecnología FPGA aplicada a la determinación de volúmenes cardíacos. Tesis de Maestría. (Tucumán: Departamento de Bioingeniería FACET UNT)

[11] Herrera M, Olivera J and Gómez López M 2010 Implantable hemodynamic monitors: can be conductance catheter system successfully implemented? 32nd Anual Int. IEEE EMBS Conf. Digital 3549-52

[12] Norma IEEE 802.15.4 2006 IEEE Standard for Telecommunications and Information Exchange Between Systems - LAN/MAN - Specific Requirements - Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks (WPANs)