System Design of RF Receiver and Digital Implementation of Control Logic

Marcus Ström

2003-04-09
System Design of RF Receiver and Digital Implementation of Control Logic

Examensarbete utfört i Elektronikdesign vid Linköpings Tekniska Högskola, Campus Norrköping

Marcus Ström

Handledare: Per-Olof Bergstedt, Magnus Sneitz
Examinator: Qin-Zhong Ye

Norrköping den 9 April 2003
System Design of RF Receiver and Digital Implementation of Control Logic

Marcus Ström

This report is the outcome of a thesis work done at Linköpings University, campus Norrköping. The thesis work was part of the development of a RF transceiver chip for implantable medical applications. The development was done in cooperation with Zarlink Semiconductor AB, located in Järfälla, Stockholm.

The transceiver is divided into three main blocks, which are the wakeup block, the MAC block and the RF block. The wakeup block is always operating and is awaiting a wakeup request in the 2,45GHz ISM-band. The RF-block is operating in the 400MHz ISM-band and is powered up after wakeup. The MAC is the controller of the whole chip. All three blocks in the transceiver structure should be integrated on the same chip, using TSMC 0,18µm process design kit for CMOS (Mixed Signal /RF).

The purpose of the thesis work was to develop the wakeup circuit for the transceiver. The main purpose was to develop the digital control logic in the circuitry, using RTL-coding (mainly VHDL) but the thesis work also included a system analysis of the whole wakeup block, including the front-end, for getting a better overview and understanding of the project.

A complete data packet or protocol for the wakeup message on 2,45GHz, is defined in the report and is one of the results of the project. The packet was developed continuously during progress in the project. Once the data packet was defined the incoming RF stage could be investigated. The final proposal to a complete system design for the wakeup block in the RF transceiver is also one of the outcomes of the project. The front-end consists mainly of a LNA, a simple detector and a special decoder. Since the total power consumption on the wakeup block was set to 200nA, this had to be taken under consideration continuously. There was an intention not to have an internal clock signal or oscillator available in the digital part (for keeping the power consumption down). The solution to this was a self-clocking method used on the incoming RF signal. A special decoder distinguishes the incoming RF signal concerning the burst lengths in time. The decoder consists of a RC net that is uploaded and then has an output of 1, if the burst length is long enough and vice versa.

When it was decided to use a LNA in the front-end, it was found that it could not be active continuously, because of the requirements on low power consumption. The solution to this was to use a strobe signal for the complete front-end, which activates it. This strobe signal was extracted in the digital logic. The strobe signal has a specific duty cycle, depending on the time factors in the detector and in the decoder in the front-end. The total strobing time is in the implemented solution 250µs every 0,5s.

The digital implementation of the control logic in the wakeup block was made in VHDL (source code) and Verilog (testbenches). The source code was synthesized against the component library for the process 0,18µm from TSMC, which is a mixed/signal and RF process. The netlist from the synthesizing was stored as a Verilog file and simulated together with the testbenches using the simulator Verilog-XL. The results from the simulations were examined and reviewed in the program Simvision from Cadence. The result was then verified during a pre-layout review together with colleagues at Zarlink Semiconductor AB. During the implementation phase a Design report was written continuously and then used for the pre-layout review. Extracts (source code and testbench) from this document can be found as appendixes to the report.

Nyckelord
Transceiver, receiver, RF, telemetry, modulation, coding, system design, digital implementation, VHDL, Verilog, RTL, low power consumption, LNA, detector and decoder.
Abstract

This report is the outcome of a thesis work done at Linköping University, campus Norrköping. The thesis work was part of the development of a RF transceiver chip for implantable medical applications. The development was done in cooperation with Zarlink Semiconductor AB, located in Järfälla, Stockholm.

The transceiver is divided into three main blocks, which are the wakeup block, the MAC block and the RF block. The wakeup block is always operating and is awaiting a wakeup request in the 2.45 GHz ISM-band. The RF-block is operating in the 400MHz ISM-band and is powered up after wakeup. The MAC is the controller of the whole chip. All three blocks in the transceiver structure should be integrated on the same chip, using TSMC 0,18µm process design kit for CMOS (Mixed Signal / RF).

The purpose of the thesis work was to develop the wakeup circuit for the transceiver. The main purpose was to develop the digital control logic in the circuitry, using RTL-coding (mainly VHDL) but the thesis work also included a system analysis of the whole wakeup block, including the front-end, in order to get a better overview and understanding of the project.

A complete data packet or protocol for the wakeup message on 2.45 GHz is defined in the report and is one of the results of the project. The packet was developed continuously during progress in the project. Once the data packet was defined the incoming RF stage could be investigated.

The final proposal to a complete system design for the wakeup block in the RF transceiver is also one of the outcomes of the project. The front-end consists mainly of an LNA, a simple detector and a special decoder. Since the total power consumption on the wakeup block was set to 200nA, this had to be taken under consideration continuously. There was an intention not to have an internal clock signal or oscillator available in the digital part (to keep the power consumption down). The solution to
Abstract

this was a self-clocking method used on the incoming RF signal. A special decoder distinguishes the incoming RF signal concerning the burst lengths in time. The decoder consists of a RC net that is uploaded and then has an output of 1, if the burst length is long enough and vice versa.

When it was decided to use an LNA in the front-end, it was found that it could not be active continuously, because of the requirements for low power consumption. The solution to this was to use a strobe signal for the complete front-end, with which it is activated. This strobe signal was extracted in the digital logic. The strobe signal has a specific duty cycle, depending on the time factors in the detector and in the decoder in the front-end. The total strobing time is in the implemented solution 250µs every 0,5s.

The digital implementation of the control logic in the wakeup block was made in VHDL (source code) and Verilog (testbenches). The source code was synthesized against the component library for the process 0,18µm from TSMC, which is a mixed signal and RF process. The netlist from the synthesizing was stored as a Verilog file and simulated together with the testbenches using the simulator Verilog-XL. The results from the simulations were examined and reviewed in the program Simvision from Cadence. The result was then verified during a pre-layout review together with colleagues at Zarlink Semiconductor AB. During the implementation phase a Design report was written continuously and then used for the pre-layout review. Extracts in form of the source code and the testbench from this document can be found as appendix B, respectively appendix C to this report.
This report is the result of the author’s thesis work performed at Zarlink Semiconductor AB in Järfälla, Stockholm. It was performed at the department Medical Applications and constitutes the final element of a Master of Science exam in Electronics Design at the University of Linköping at Campus Norrköping.

The work was performed at Zarlink Semiconductor AB during the autumn and winter 2002-2003, under supervision of Per-Olof Bergstedt and Magnus Sneitz. I really want to thank these people plus Tony Ohlsson at the company and of course my examiner Qin-Zhong Ye at the Institution of Science and Technology (ITN) at Campus Norrköping. Without their support this work would never have reached this final result.
# Contents

1 **GENERAL INTRODUCTION** .......................................................................................................................... 9  
1.1 PURPOSE & METHOD .......................................................................................................................... 10  
1.2 STRUCTURE OF THE REPORT ................................................................................................. 10  

2 **THEORY** ............................................................................................................................................ 13  
2.1 WHAT IS TELEMETRY? .................................................................................................................. 13  
2.2 RADIO COMMUNICATION ............................................................................................................ 14  
2.2.1 Modulation techniques ........................................................................................................ 14  
2.2.2 Demodulation & detection .................................................................................................. 19  
2.2.3 Frequency regulations ....................................................................................................... 20  
2.3 CODING & ERROR DETECTION .................................................................................................... 20  
2.3.1 Manchester encoding ........................................................................................................ 20  
2.3.2 Parity check ....................................................................................................................... 21  
2.3.3 Checksum .......................................................................................................................... 21  
2.3.4 CRC ................................................................................................................................. 22  

3 **SYSTEM ANALYSIS** ...................................................................................................................... 23  
3.1 THE DATA PACKET ..................................................................................................................... 24  
3.2 MODULATION & DEMODULATION .......................................................................................... 27  
3.3 THE DECODER .......................................................................................................................... 28  
3.4 TIME REQUIREMENTS ON SENT DATA .................................................................................. 28  
3.5 ATTENUATION OF RF SIGNAL .................................................................................................. 29  
3.6 SYSTEM DESIGN ......................................................................................................................... 30  
3.6.1 General description ............................................................................................................ 30  
3.6.2 Pin description .................................................................................................................. 32  
3.6.3 The LNA – wake_lna ...................................................................................................... 33  
3.6.4 The detector – wake_det ................................................................................................. 33  
3.6.5 The decoder – wake_dec ............................................................................................... 34  
3.6.6 The digital block - wake_ctrl ....................................................................................... 35  

---

**Table of contents**

1. **GENERAL INTRODUCTION**  
   1.1 PURPOSE & METHOD  
   1.2 STRUCTURE OF THE REPORT  

2. **THEORY**  
   2.1 WHAT IS TELEMETRY?  
   2.2 RADIO COMMUNICATION  
   2.2.1 Modulation techniques  
   2.2.2 Demodulation & detection  
   2.2.3 Frequency regulations  
   2.3 CODING & ERROR DETECTION  
   2.3.1 Manchester encoding  
   2.3.2 Parity check  
   2.3.3 Checksum  
   2.3.4 CRC  

3. **SYSTEM ANALYSIS**  
   3.1 THE DATA PACKET  
   3.2 MODULATION & DEMODULATION  
   3.3 THE DECODER  
   3.4 TIME REQUIREMENTS ON SENT DATA  
   3.5 ATTENUATION OF RF SIGNAL  
   3.6 SYSTEM DESIGN  
   3.6.1 General description  
   3.6.2 Pin description  
   3.6.3 The LNA – wake_lna  
   3.6.4 The detector – wake_det  
   3.6.5 The decoder – wake_dec  
   3.6.6 The digital block - wake_ctrl
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.7</td>
<td>THE STROBING TIME</td>
<td>37</td>
</tr>
<tr>
<td>3.7.1</td>
<td>Solution A</td>
<td>38</td>
</tr>
<tr>
<td>3.7.2</td>
<td>Solution B</td>
<td>38</td>
</tr>
<tr>
<td>4</td>
<td>DIGITAL IMPLEMENTATION</td>
<td>39</td>
</tr>
<tr>
<td>4.1</td>
<td>TOP LEVEL – WAKE_CTRL_A</td>
<td>40</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Sub block start_check_a</td>
<td>42</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Sub block wakein_decode_a</td>
<td>42</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Sub block para_data_inout_a</td>
<td>43</td>
</tr>
<tr>
<td>4.1.4</td>
<td>Sub block clk_div_a</td>
<td>44</td>
</tr>
<tr>
<td>4.2</td>
<td>SIMULATIONS</td>
<td>45</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Top simulation</td>
<td>45</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Simulation close-ups</td>
<td>46</td>
</tr>
<tr>
<td>4.3</td>
<td>DESIGN PROCESS</td>
<td>50</td>
</tr>
<tr>
<td>4.3.1</td>
<td>RTL-coding</td>
<td>50</td>
</tr>
<tr>
<td>4.3.2</td>
<td>RTL simulation</td>
<td>51</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Synthesis</td>
<td>51</td>
</tr>
<tr>
<td>4.3.4</td>
<td>Pre-layout verification</td>
<td>52</td>
</tr>
<tr>
<td>4.3.5</td>
<td>Layout</td>
<td>52</td>
</tr>
<tr>
<td>4.3.6</td>
<td>Post layout verification</td>
<td>52</td>
</tr>
<tr>
<td>5</td>
<td>RESULTS</td>
<td>55</td>
</tr>
<tr>
<td>5.1</td>
<td>FUTURE IMPROVEMENTS</td>
<td>56</td>
</tr>
<tr>
<td>6</td>
<td>REFERENCES</td>
<td>57</td>
</tr>
<tr>
<td>APPENDIX A</td>
<td>ABBREVIATIONS</td>
<td>61</td>
</tr>
<tr>
<td>APPENDIX B</td>
<td>SOURCE CODE</td>
<td>63</td>
</tr>
<tr>
<td>APPENDIX C</td>
<td>TESTBENCH</td>
<td>73</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>FIGURE 1</td>
<td>VIEW OVER THE RF TRANSCEIVER CHIP ENVIRONMENT</td>
<td>9</td>
</tr>
<tr>
<td>FIGURE 2</td>
<td>EXAMPLE OF TELEMETRY SYSTEM WITH A RF-LINK IMPLEMENTED</td>
<td>14</td>
</tr>
<tr>
<td>FIGURE 3</td>
<td>SIGNAL CHARACTERISTICS TO MODIFY</td>
<td>15</td>
</tr>
<tr>
<td>FIGURE 4</td>
<td>COMPARISON BETWEEN BASIC MODULATION TECHNIQUES</td>
<td>15</td>
</tr>
<tr>
<td>FIGURE 5</td>
<td>SIGNAL DIAGRAMS FOR OOK AND ASK</td>
<td>16</td>
</tr>
<tr>
<td>FIGURE 6</td>
<td>BASK IN ITS SIMPLEST FORM, ON/OFF KEYING (OOK)</td>
<td>16</td>
</tr>
<tr>
<td>FIGURE 7</td>
<td>BINARY FREQUENCY-SHIFT KEYING</td>
<td>17</td>
</tr>
<tr>
<td>FIGURE 8</td>
<td>BINARY PHASE-SHIFT KEYING (BPSK)</td>
<td>18</td>
</tr>
<tr>
<td>FIGURE 9</td>
<td>SIGNAL DIAGRAM FOR BPSK AND QPSK</td>
<td>18</td>
</tr>
<tr>
<td>FIGURE 10</td>
<td>COMPARISON BETWEEN BPSK AND QPSK</td>
<td>19</td>
</tr>
<tr>
<td>FIGURE 11</td>
<td>PRINCIPLE OF MANCHESTER ENCODING</td>
<td>20</td>
</tr>
<tr>
<td>FIGURE 12</td>
<td>PRINCIPLE OF DIFFERENTIAL MANCHESTER</td>
<td>21</td>
</tr>
<tr>
<td>FIGURE 13</td>
<td>THE FIRST APPROACH OF THE DATA PACKET</td>
<td>25</td>
</tr>
<tr>
<td>FIGURE 14</td>
<td>THE SECOND DATA PACKET</td>
<td>26</td>
</tr>
<tr>
<td>FIGURE 15</td>
<td>THE FINAL AND DEFINITE DATA PACKET</td>
<td>27</td>
</tr>
<tr>
<td>FIGURE 16</td>
<td>SCHEMATIC OVERVIEW OF A SIMPLE DETECTOR STAGE</td>
<td>28</td>
</tr>
<tr>
<td>FIGURE 17</td>
<td>SCHEMATIC DESCRIBING A SL PACKET</td>
<td>29</td>
</tr>
<tr>
<td>FIGURE 18</td>
<td>SCHEMATIC OVERVIEW OF THE WAKEUP CIRCUITRY</td>
<td>31</td>
</tr>
<tr>
<td>FIGURE 19</td>
<td>SCHEMATIC OVER THE DECODER – THE WAKE_DEC</td>
<td>34</td>
</tr>
<tr>
<td>FIGURE 20</td>
<td>EXPLANATION OF THE DECODING OF THE SIGNAL IN THE DECODER</td>
<td>34</td>
</tr>
<tr>
<td>FIGURE 21</td>
<td>TIME REQUIREMENTS OF THE RC NET IN THE DECODER</td>
<td>35</td>
</tr>
<tr>
<td>FIGURE 22</td>
<td>SCHEMATIC OVER THE DIGITAL BLOCK – WAKE_CTRL</td>
<td>36</td>
</tr>
<tr>
<td>FIGURE 23</td>
<td>TOP LEVEL OF THE SYNTHESIZED IMPLEMENTATION</td>
<td>40</td>
</tr>
<tr>
<td>FIGURE 24</td>
<td>TOP SIMULATION WITH CORRECT TESTBENCH</td>
<td>45</td>
</tr>
<tr>
<td>FIGURE 25</td>
<td>WRITE-TO-REGISTER FUNCTION VIA THE BUS SYSTEM</td>
<td>46</td>
</tr>
<tr>
<td>FIGURE 26</td>
<td>READ-FROM-REGISTER FUNCTION VIA THE BUS SYSTEM</td>
<td>46</td>
</tr>
<tr>
<td>FIGURE 27</td>
<td>DECODING OF START SEQUENCE AND PROLONGING OF STROBE SIGNAL</td>
<td>47</td>
</tr>
<tr>
<td>FIGURE 28</td>
<td>DECODING OF THE COMPANY CODE AND APPLICATION ID</td>
<td>47</td>
</tr>
<tr>
<td>FIGURE 29</td>
<td>RESET OF LOGIC WHEN WRONG CODE IS INCOMING (MANCHESTER CODE)</td>
<td>48</td>
</tr>
<tr>
<td>FIGURE 30</td>
<td>RESET OF LOGIC WHEN WRONG CODE IS COMING IN (APPLICATION ID)</td>
<td>48</td>
</tr>
<tr>
<td>FIGURE 31</td>
<td>STROBING CYCLE AND THE FUNCTION OF THE WU_EN SIGNAL</td>
<td>48</td>
</tr>
<tr>
<td>FIGURE 32</td>
<td>FUNCTION OF THE IBS_IN SIGNAL</td>
<td>49</td>
</tr>
<tr>
<td>FIGURE 33</td>
<td>DIGITAL BLOCK DESIGN PROCESS</td>
<td>50</td>
</tr>
</tbody>
</table>
This report is the outcome of a thesis work done at Linköping University. The thesis work was part of the development of a RF transceiver chip for implantable medical applications. The development was done in cooperation with Zarlink Semiconductor AB, located in Järfälla, Stockholm. The system architecture was predefined and can be described as below in Figure 1.

**Figure 1 View over the RF transceiver chip environment**

The RF Transceiver should initially be designed for implantable applications. However, the development should also target a design for a generic technology platform for future spin-off applications within the frequency band 300 - 1000 MHz.
The main markets are products in the area of telemetry and implantable medical applications.

The complete system will consist of a base station that communicates with one or several implants as seen in Figure 1. The base station consists of a 2.45 GHz antenna (A3), an 400 MHz antenna (A2), a RF PCB with the RF transceiver chip bonded to it and a 2.45 GHz transmitter circuitry attached. An application PCB with a base station chip or a direct interface to a PC or PDA and a battery power supply or feed from PC or PDA is also needed. The implant consists of an RF PCB with the same RF transceiver chip, a combined 400MHz/ 2450MHz patch antenna, the implant application chip and a battery power supply.

The RF Transceiver chip shall be configured for operation on the implant side but also for the base station side. The transceiver is divided into three main blocks, which are the wakeup block, the MAC block and the RF block. The wakeup block is always operating and is awaiting a wakeup request in the 2.45 GHz ISM-band (this function disconnected when operating as a base station). The RF-block is operating in the 400MHz ISM-band and is powered up after wakeup or directly in the base station application. The MAC is the controller of the whole chip and if serving as a base station also controlling the external 2.45 GHz transmitter. All three blocks in the transceiver structure should be integrated on the same chip, using TSMC 0,18µm process design kit for CMOS (Mixed Signal / RF).

1.1 Purpose & method

The purpose of the thesis work was to develop the wakeup circuit for the RF transceiver. The main purpose was to get familiar with digital implementation using RTL-coding (mainly VHDL) but the thesis work also included a system analysis of the whole wakeup block, including the front-end, in order to get a better overview and understanding of the project. The source code for the design was written in VHDL and then synthesized using the synthesis tool Design Vision from Synopsys. The output from the synthesis tool, the netlist, was stored as Verilog file and simulated in Verilog XL with testbenches written in Verilog.

During the thesis work a requirement specification was written for the complete wakeup block and a design report was written for the digital implementation. These documents were done for documentation purpose at Zarlink Semiconductor AB. Most of the contents of these documents are found in the report but some parts were confidential and were therefore excluded. The design report included the source code and the testbenches, which are found in this report as appendixes.

1.2 Structure of the report

The report is divided into three main parts. The first part is mainly theoretical and is introducing some different, adequate terms and concepts for better understanding of
General Introduction

the report. This part is mainly dealing with terms like telemetry and radio communication. Within these areas there are parts like modulation, demodulation, error detection and coding techniques. This part is necessary as an introduction to the area of low power and RF design.

The second part is the system analysis, which describes the development of the wakeup block. It describes some of the problems that came up during the research phase and describes and motivates the different choices made. Continuously during the work, new information and approaches were discussed, mainly with people at Zarlink Semiconductor.

In the final part, the implementation, the work with the RTL-coding, simulation and verification are described and presented. The actual implementation work is described and each sub block in the design are presented and described at a behavioral level. The simulation procedure is described and extracts showing simulations results are presented.

The actual RTL-code (in VHDL) is found in Appendix B and the testbench (in Verilog) is found in Appendix C. Appendix A is just a compilation of all abbreviations used in the report and is most functional to use as a reference list.
2 Theory

The theory chapter describes relevant and adequate information for the project, which can be useful to read for better understanding of the rest of the report. If the reader already is familiar with terms like telemetry, radio communication, modulation techniques (including FSK, PSK and ASK) and error coding techniques like Manchester encoding, Parity check and CRC it is not necessary to read this chapter.

2.1 What is telemetry?

Telemetry is the term for using wireless radio communication for sending automatic indications or manually read values from different measuring instrument or devices in for example hospital environments. The usage of this technique is becoming more and more common and new usage areas are evolving. If it would be possible to implement some kind of low power and high speed RF transceiver in different medical applications, this solution would have many benefits and new usage areas against today’s techniques.

Today’s system, in pacemaker for example, uses a low speed, very low distance inductive communication system, that is limited to data rates of a few kilobits per second at a decimeter range. An integrated RF-link on the other hand can offer much higher data rates and communication distances up to some meters. If high data rates are available together with applications like these, then there is a possibility to easily download for example data statistics from a patient. Other usage areas could be for example body-implanted sensors, which can communicate with each other over a wireless RF transceiver solution.
The development within radio communication using digital transmission leads towards safer built systems with fewer errors in the transmission. When using digital transmission the possibilities to control the data flow increases and, if wanted, include correction of the transmitted data, which decreases the total amount of errors. [2], [5]

2.2 Radio communication

The transmission of digital signals is increasing around the world at a rapid rate and the number of users at different frequencies is quickly increasing. The source signals in radio communication are often referred to as base band signals. From electromagnetic theory is that for efficient radiation of electrical energy from an antenna it must be at least in the order of the magnitude of a wavelength in the size $c = f \cdot \lambda$, where $c$ is the velocity of light, $f$ is the signal frequency and $\lambda$ is the wavelength. Then for sending a base band signal wireless, it is often frequency-translated to a higher frequency range for efficient transmission. This process is called modulation. Digital modulation provides more information capacity, compatibility with digital data services, higher data security and many more benefits towards analogue modulation.

For transmitting a signal over the air, there are three main steps. First a pure carrier is generated at the transmitter. In the modulation process the base band signal constitutes the modulating signal and the higher frequency, the carriers signal as a sinusoidal waveform. This modulation type is also sometimes called carrier wave (CW) modulation. The following chapters will describe the most basic digital modulation techniques, Amplitude-shift keying (ASK), Frequency-shift keying (FSK) and Phase-shift keying (PSK). [2], [5], [6], [27], [28], [29], [30], [31]

2.2.1 Modulation techniques

There are only three characteristics of a signal that can be changed over time and these are amplitude, phase and frequency. However, phase and frequency are just different
ways to view or measure the same signal change. Amplitude and phase can also be modulated simultaneously but this is difficult to generate and even more difficult to detect.

Figure 3 Signal Characteristics to Modify

The following chapters will only describe binary digital modulation techniques in detail. This is done only to keep this theory as adequate to the project as possible and as an introduction to these different techniques. There are of course other modulation techniques, like I/Q modulation etc. These techniques are just mentioned briefly under each chapter. The thesis work was concentrated to (ultra) low power requirements and therefore more complicated modulation techniques were unnecessary to investigate. This because the more complicated the modulation is, the more complicated the architecture of the transmitter and the receiver becomes, which means more power consumption in reality. [1], [3], [4], [27], [28], [29], [30], [31]

Figure 4 Comparison between basic modulation techniques
**Amplitude-Shift Keying (ASK)**

The principle of Amplitude-Shift Keying (ASK) is very simple. Mathematically it is equivalent to multiplying the carrier signal by the binary data signal (or specific magnitude amplitude representing each logic level). A Binary ASK (BASK) is defined by:

\[ s(t) = A \cdot m(t) \cdot \cos(2\pi f_c t), \quad 0 \leq t \leq T \]

Where \( A \) is a constant, \( m(t) = 1 \) or \( 0 \), \( f_c \) is the carrier frequency and \( T \) is the bit duration. In the literature this modulation technique is sometimes divided into two types. These are the ordinary ASK and the On/Off Keying (OOK), which is a special case of ASK. These two modulation types can be graphically represented on a two dimensional ortho-normal plot, sometimes referred to as a signal diagram. This is done in order to get a better understanding and therefore the signal diagrams for OOK and ASK is shown in Figure 5.

![Signal diagrams for OOK and ASK](image)

**Figure 5 Signal diagrams for OOK and ASK**

On/off keying (OOK) is the simplest ASK modulation where the transmission is either on/off to represent 1/0. Hence, it is the special case of ASK modulation where no carrier is present during the transmission of a zero. It is a non-coherent modulation technique that represents a good compromise between power efficiency and bandwidth efficiency and lends itself to a simple receiver structure. [1], [3], [4], [27], [28], [29]

![BASK in its simplest form, On/Off keying (OOK)](image)

**Figure 6 BASK in its simplest form, On/Off keying (OOK)**
Frequency-Shift Keying (FSK)

In Frequency-Shift Keying (FSK), the frequency of the carrier wave is changed as a function of the modulating signal (the data). Amplitude remains unchanged in this technique. In the simplest case, Binary FSK (BFSK), a ‘1’ is represented by one frequency and a ‘0’ is represented by another frequency. The difference between the mark and space frequencies, called the shift, is usually between 100 and 1000Hz. A BFSK signal can be defined by

\[
s(t) = \begin{cases} 
A \cos 2\pi f_0 t, & 0 \leq t \leq T \\
A \cos 2\pi f_1 t, & \text{elsewhere}
\end{cases}
\]

Where \( A \) is a constant, \( f_0 \) and \( f_1 \) are the transmitted frequencies and \( T \) is the bit duration.

![Figure 7 Binary Frequency-Shift Keying](image)

Phase-Shift Keying (PSK)

Phase-shift keying (PSK) involves shifting the phase of the carrier to represent digits. In PSK, the binary signal (0 or 1) to be transmitted changes the phase shift of the sine wave carrier accordingly. The simplest form of PSK is known as Binary PSK (BPSK). When a binary 0 occurs, the carrier signal is transmitted with one phase, but when binary 1 occurs, the carrier signal is transmitted with 180° phase shift. Hence, in BPSK the phase of the carrier is modulated according to the data that should be transmitted as followed:

- When binary 1, \( \phi = 0 \)
- When binary 0, \( \phi = \pi \) (180°)
The main problem with BPSK is that the speed of data transmission is limited in a given bandwidth. One way to increase the binary data rate without increasing the bandwidth requirement is to encode more than one bit per phase change. The simplest way to do this is Quadrature PSK (QPSK), where each symbol represents 2 bits.

Quadrature Phase Shift Keying (QPSK) is used extensively in applications including CDMA (Code Division Multiple Access) cellular service, wireless applications, Iridium (a voice/data satellite system) and DVB-S (Digital Video Broadcasting - Satellite). Quadrature means that the signal shifts between phase states which are separated by 90 degrees (instead of 180 degrees as in BPSK). The signal shifts in increments of 90 degrees from 45 to 135, –45, or –135 degrees. These points are chosen as they can be easily implemented using I/Q modulator. This allows two discrete data streams, identified as I channel (in phase) and Q channel (quadrature) data. Only two I values and two Q values are needed and this gives two bits per symbol. There are four states because $2^2 = 4$. For those reason this is a more bandwidth-efficient type of modulation than BPSK, potentially twice as efficient.
To achieve greater efficiencies the number of bits per symbol must be increased even more. If QPSK is generalized to more than four constellation points, it is known as M-PSK, where M represents the number of constellation points. M is a power of 2 greater than or equal to 8. Because of this increased bandwidth the price is reduced power efficiency. [1], [3], [4], [27], [28], [29], [30]

![Diagram of BPSK and QPSK](image)

**Figure 10 Comparison between BPSK and QPSK**

### 2.2.2 Demodulation & detection

The demodulator or the detector is the circuit, in which the recreation of the original modulating frequency (the information) from the carrier frequency is done. The output of an ideal detector must be an exact reproduction of the modulation existing on the carrier wave. Failure to accurately recover this intelligence will result in distortion and degradation of the demodulated signal and perhaps some information will be lost.

Carrier modulation allows the transmission of modulating frequencies without the use of transmission wire as mentioned before. However, for the communication process to be completed or to be useful, the information or data must be recovered in its original form at the receiving site. Each type of modulation is different and requires different techniques to recover (demodulate) the information.

To detect and recover an OOK (On/Off Keyed) signal, a method of detecting the presence or absence of RF oscillation is necessary. Demodulators that detect the presence of RF oscillations and convert them into a recognizable form have a non-complicated architecture and are therefore useful in low power constructions. [1], [3], [5], [7], [8], [9]

**Coherent and non-coherent**

In theory, demodulation differs between coherent and non-coherent systems. In coherent systems the receiver knows the exact phase and frequency of the carrier at all times. On the other hand, non-coherent systems mean that the receiver must obtain the phase and frequency of the carrier from the received signal. [4], [7]
2.2.3 Frequency regulations

The frequency that is used for the wakeup message is 2.45 GHz. According to the Swedish PTS (Post & Tele Styrelsen), 2 400–2 500 MHz (center frequency 2450 MHz) are designated for Industrial, Scientific and Medical (ISM) applications. Radio communication services operating within these bands must accept harmful interference, which may be caused by these applications. According to updated standards and regulations please visit www.etsi.org for information.

ETSI (the European Telecommunications Standards Institute) is an organization whose mission is to produce the telecommunications standards that will be used throughout Europe and beyond. ETSI plays a major role in developing a wide range of standards and other technical documentation as Europe's contribution to worldwide standardization in telecommunications, broadcasting and information technology. The prime objective of ETSI is to support global harmonization by providing a forum in which all the key players can contribute actively. The European Commission and the EFTA secretariat officially recognize ETSI. [25], [26]

2.3 Coding & error detection

When using wireless radio communication there are several different types of error detection techniques. In this subsection some of these techniques will be described briefly. For radio communications over wireless media, Manchester encoding is very often used as error detection.

2.3.1 Manchester encoding

Manchester encoding is a type of error detection that is often used by different RF devices. It is a coding that represents an original bit by another set of bits. If the original data is a Logic 0, the Manchester code is 0 to 1 (upward transition at bit center) and if the original data is a Logic 1, the Manchester code is 1 to 0 (downward transition at bit center), read from right to left in Figure 11.

<table>
<thead>
<tr>
<th>Binary Data</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manchester</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 11 Principle of Manchester encoding**

It can be seen that there are two bits of Manchester encoded data for each bit of original data. The penalty for doing this is, is that Manchester encoded data consumes double bandwidth. The coding can be seen as a synchronous clock encoding technique used to encode the clock and data of a synchronous bit stream. The encoding may also be alternatively viewed as a phase encoding where each bit is encoded by a positive 90
degree phase transition, or a negative 90 degree phase transition. The Manchester code is therefore sometimes also known as a bi-phase code.

In general, when transmitting serial data to a radio receiver, a DC component of zero must be maintained (over a finite time). This is so the demodulator in the receiver can properly interpret the received data as 1's and 0's. Manchester encoding allows to do this, because either 01 or 10 always represent each bit.

Given that a 0 encodes 10 and a 1 encodes 01, it follows that 00 and 11 are illegal sequences or codes during the data information. These codes are sometimes used to error check the data. It is possible to have the 8-bit not Manchester encoded “illegal code” 11110000, which is an unlikely occurrence. This “illegal code” has the property of having a DC component of zero and has one, 0 to 1 transition. This code can therefore be used as a unique start sequence identifying the boundaries of a Manchester encoded bit stream or data frame.

![Figure 12 Principle of Differential Manchester](image)

The mid-bit transition in Manchester code can serve as a clock as well as data. A low to high transition represents a 1 and a high to low transition represents a 0. In so called Differential Manchester (Figure 12) the mid-bit transition is used only to provide clocking. Encoding of a 0 is represented by a transition at the beginning of a bit period and a 1 is represented by the absence of a transition at the beginning of a bit period. [1], [3], [7], [12], [13], [14], [23]

2.3.2 Parity check

Parity check is one of the simplest techniques of error detection. Parity check always means either even or odd parity, where even parity means that the total number of 1’s must be even and vice versa for odd parity. The parity bit is added in a data packet as an extra bit and this technique is usually used in small data packets. In reality the calculation is done with XOR of all bits in the packet. Parity check is good for single bit errors and if odd numbers of bits are changed during the data transfer. If even numbers of bits are changed in the packet it is undetectable. [1], [15], [16], [23]

2.3.3 Checksum

Checksum is a traditional way of error detection. Checksum is using one’s complement arithmetic for summation of the data packet. The sender calculates the checksum (summation of all data blocks plus adding of a 1) and sends the one’s complement of the checksum together with the original data. The receiver adds all
incoming blocks as it receives them and adds a 1 at the end. If the total sum is not 0, the receiver has detected an error. Note that sometimes the term checksum is used generically to mean error detection for many methods. [1], [3], [15], [16], [23]

2.3.4 CRC

CRC or Cyclic Redundant Check is a strong method for detecting errors in data communication. It is a polynomial that calculates a checksum for the data packet to be sent. The transmitter sends this checksum at the end of the block and the receiver does the same calculation of the incoming data with the same polynomial and compares this with the incoming checksum. Depending on which polynomial chosen, the method is more or less secure. CRC is used because it is very useful in detecting single bit errors, multiple bit errors, and burst errors in data messages. An example of a CRC is CRC-16, which uses the polynomial \( G_x = X^{16} + X^{15} + X^2 + 1 \) (This polynomial is of degree 16). [3], [17], [18], [19], [20], [23]
3 System Analysis

The system analysis of the wakeup block in the RF transceiver was one of the two main parts of the thesis work. This block/system was not defined at all and there were many things to examine. One of the first things to investigate was how the data packet, which should be sent as a wakeup signal on 2.45 GHz from a base station to an implant, could be defined. Since the wakeup block is intended to be active during a long time the requirement on total power consumption was set to maximum 200nA.

The incoming RF stage (also called front-end) needed to be investigated. For example, in the beginning there was an approach of not having a low noise amplifier (LNA) in this block, because of the low power requirements. This was later found to be impossible, because the signal that was sent on the system was too weak for detection without an LNA in the front-end. A question that came up was, how could this be implemented and still have the same low power consumption?

More questions about how the awakening of the rest of the circuitry should be done were also investigated. In the rest of the chip there are both analogue parts and digital parts, which have different supply levels (vdd). Regulators for achieving these supply levels must be implemented in the wakeup.

Early in the analysis part a requirement for not having an internal clock source in the wakeup was determined. This requirement forced the wakeup block to include a special solution with a decoder, which can extract a clock signal from the incoming RF signal and use this as a self-clocking method. Later in the project, it was found that it was necessary to have a clock signal for calculating the strobe time for the strobing of the analogue front-end for keeping the power consumption as low as required. Despite this, the external clock signal was decided not to be used for other purposes in the digital logic, because in future products the strobe function may be excluded and then the clock signal will be unnecessary and can be removed.
3.1 The data packet

All data that should be transmitted to the wakeup circuit for detection must be specified so both the transmitter and the receiver knows what to do. A data packet had to be defined, so the transmitter knows what data the receiver wants to receive and in which order. Some of the first questions that were necessary to look into in the beginning were these:

- What information and how much data must be sent from the transmitter to the wakeup circuitry?
- What is necessary/desirable to include in the wakeup data packet?
- How will the receiver detect the RF signal with as low power consumption as possible?
- What information and how much data are necessary to include in the data packet to have a secure and safe wakeup of the circuitry?
- What kind of modulation/demodulation will be the most suitable for the requirements on the circuitry and the RF-link?
- How will the transmitter distinguish between which receiver it wants to awake?

Out of these questions, an examination of how the data packet should be constructed was done. There has been an extensive discussion about these questions during the analysis phase of the thesis work.

The first approach was that in a data packet there must be some kind of start bits or synchronization bits so the receiver to know that what is coming in is information and not noise. The data that is sent to the receiver must also in this case contain some kind of identification bits so the receiver knows that the transmitter wants to awake the specific receiver, which receives the present RF signal at the moment.

The main purpose of the wakeup procedure was determined as this. The wakeup block receives a code that provides information about on which channel the RF circuitry shall initiate contact with the base station. The RF block shall answer on the 400MHz ISM-band, which is divided into 10 different channels and therefore the implant needs to know on which channel the base station is listening on. A first data packet was defined (Figure 13):
The first approach was a kind of a standard structure of a data packet. It included a preamble (start bits) for making it easier for the receiver to understand that a data packet is on the way in. The preamble was later found unnecessary because this receiver do not need these extra bits as a startup. The 24 synchronization bits were also found unnecessary because it would be enough with a few bits as a unique start sequence. When it later was decided to use Manchester encoding on the data, there was an opportunity to have a unique start sequence that during data transmission can not occur. Because of this a special pattern of 8 bits was created and replaced the old preamble and synchronization bits. The approach with the application ID (26bits) was used, but it was extended with a company code (8bits). This was done because there was a need for the possibility to have more different ID’s for the applications inside a specific company (which has the same company code).

With the extension of the number of bits for ID’s, from 26 bits to a total of 34 bits, the number of ID’s increased from $2^{26} = 67108864$ to $2^{34} = 17179869184$, which was more satisfactory and suitable for the worldwide area of use. The Channel ID remained in the data packet but was sent twice instead of having a checksum or CRC. This decision was made because all bits without these 4 bits are checked against registers with the correct bits. A CRC or checksum would therefore be superfluous and would work as a double check, which only consumes power and time. The sum up was that the checksum (or CRC) was removed and replaced by another transmission of the channel ID. These two (4 bits long) channel ID’s are then compared to each other and therefore checked against incorrectness. A new and second data packet was settled, which included Manchester encoding (Figure 14).
In order to get a secure and safe startup of the communication; the packet should start with a unique start sequence. These bits are necessary and facilitate for the receiver to understand that a packet is on the way. The start bits are in this case included in the total data packet. Manchester encoding was introduced for the opportunity to have a unique start sequence and a good error detection. The pattern 11110000 can never appear during data transfer because of the Manchester encoding (see chapter Manchester encoding on page 20). Hence, the start sequence is sent without Manchester encoding and is therefore unique, compared to the rest of the sent data, which are Manchester encoded.

After the start sequence, the actual information or data is sent. The data is transmitted with Manchester encoding, as mentioned above; therefore the numbers of sent bits are doubled for the data. This is where the information about which device the base station wants to wakeup is sent. A company code, 8 bits long, and an application ID of 26 bits identify this. When the correct company code and the correct application ID are detected in the wakeup, it powers up the rest of the circuitry.

Later, during the analysis phase some details about the data packet were changed and clarified. The first thing was the last bits, the channel ID. These bits were increased for the opportunity of selecting more channels. This action was mainly settled for future spin-off products to the design. It was also prolonged with 4 bits for band selection. Totally the channel ID was increased to 12 bits and it was found that there was no need for extra error detection on these bits in addition to the Manchester code already applied.

The final definition of the data packet were also increased with the calculation of the total amount of transmitted bits according to a special pattern consisting of short and long burst length (for more information about this, see chapter The decoder on page 28). The final data packet definition is found in Figure 15 below. [1], [3], [4], [7], [12], [13], [14]
When the data packet was defined, an investigation about different modulation techniques was done. The three main modulation techniques Amplitude Shift-Keying (ASK), Frequency Shift-Keying (FSK) and Phase Shift-Keying (PSK) were examined (see chapter Modulation techniques on page 14 for more information about these techniques). When the number of bits in the data packet was defined it was obvious that no complicated modulation was needed. The small amount of bits meant that the data speed on the RF link did not have to be that high for keeping the total transmission time low. The first approach was that it would be enough to have binary modulation technique and no I/Q modulation or any other more complicated technique.

The low power requirements on the receiver also made it hard to have a complicated architecture on the detector. If the modulation is complicated and complex the architecture of the receiver becomes more complex and with that comes more power consumption. After examination of Binary ASK (BASK), Binary FSK (BFSK) and Binary PSK (BPSK), it was found that the most non-complicated demodulation would be the special case of BASK called On/Off Keying (OOK). This technique is the simplest ASK modulation. The data transmission is either on/off (at 2.45 GHz in this case) for representing 1/0. The demodulation or detection of OOK is very basic; the receiver just needs to detect the presence or absence of the oscillation. Demodulators that do this and convert the incoming signal into recognizable form are most suitable for low power requirements on receivers. The architecture of this kind of detector is very simple and can be described by the basic schematic in Figure 16.
The schematic above includes a diode detector plus a comparator. The principle is that the capacitor charges through the diode during positive half-cycles of the carrier for binary data ‘1’, and discharges through the resistor for data ‘0’. The figure is an extract from the requirement specification done for the wakeup block during the thesis work. [2], [5], [7], [8], [9], [21], [22]

3.3 The decoder

As mentioned in the beginning of the system analysis chapter, there were in the beginning an approach of not having an internal clock or oscillator in the wakeup block. The main reason was the requirement on low power consumption. This forced the wakeup block to include a special solution with a decoder, which can extract a clock signal from the incoming RF signal and use this as a sort of self-clocking method.

The solution to this problem was to implement a circuitry that can distinguish two specified patterns from each other, depending on the time between the negative pulse edges. The decoder may consist of an RC net that can distinguish these signals. The incoming signal to the block consists of either short 0 followed by a short 1 or long 0 followed by long 1. Short and long are the time factors for the absence/presence of the RF signal.

The data that flow into a block like this must consist of an OOK modulated signal. The incoming signal shall also be used as the clock for its output. Hence, the incoming signal is the clock signal to the output when these two flow into the digital circuitry. A solution like this was examined and later used because of the requirements of, on low power consumption. For viewpoint of power consumption, there was no possibility for internal clock synchronization in the wakeup block as mentioned.

3.4 Time requirements on sent data

The final data packet is total 54 bits without Manchester encoding and the encoding for the special decoder mentioned above. If the Manchester encoding is added and the amount of bits is doubled according to the special decoder used, the total amount of transmitted bits will be 200 (the start sequence of 8 bits is not Manchester encoded). The total data packet that will be sent is accordingly 200 bits (see Figure 15 on page 27) or 50 packets of either SL or LS packets. A transmitted SL packet consists of a
short 0 followed by a short 1 followed by a long 0 and finally followed by a long 1 (See Figure 17 below). A SL packet is translated to an output of a 0 followed by a 1 in the special decoder. This data is then decoded according to Manchester code to a 1 in the digital block. From this follows that 4 transmitted bits only represents 1 bit in the digital logic.

Figure 17 Schematic describing a SL packet

The time for an SL or LS packet to be sent is 0.5Xs (short 0) + 0.5Xs (short 1) + 1.5Xs (long 0) + 1.5Xs (long 1) = 4Xs. The time 0.5X is a critical minimum time for the comparator in the detector to let the signal pass through. The time 3Xs is the critical time for the RC net in the special decoder to be able to let through a signal as a 1. If the time is long enough to upload the RC net in the decoder it will put out a 1, otherwise the signal into the digital part will be clocked as a 0. The time for an SL packet to be sent is therefore 50 * 4Xs = 200Xs, plus some delay time. The total time will be the same for an LS packet.

3.5 Attenuation of RF signal

During the thesis work there was a big risk with the implantable RF-transceiver, using 2.45 GHz as the operating frequency for the wakeup message. How much will the RF signal attenuate in free space, and how will it behave concerning the attenuation in human tissues inside a body? Questions like this needed to be examined.

Since the issue with attenuation in body tissue was considered a big risk to the project, it was decided that it was necessary to settle a field measurement for this. The purpose of this measurement was to define the body attenuation of a 2.45 GHz signal. Effects of different implantation depth were to be evaluated.

The result from the field measurements showed that the attenuation in human body tissues at 2.45 GHz is about 7.9dB/22mm, which verifies the information the company got from an internal source. Since the maximum output power on 2.45 GHz is set to 100mW, and the free space loss (on 2.45 GHz) at 2m distance is approximately 46dB. Free space loss is actually the amount of attenuation of RF energy on an unobstructed path between two isotropic antennas. Basically, it is the dilution of energy as the RF
signal propagates away from a source. The free space loss can be calculated from the equation below (EQ 1).

$$\text{Free Space Loss} = 20 \log_{10} (\text{Frequency in MHz}) + 20 \log_{10} (\text{Distance in Miles}) + 36.6 \quad (EQ 1)$$

At 2 meters (0.001243 miles) distance the free space loss (attenuation) for 2.45 GHz is:

$$20 \log_{10} (2450) + 20 \log_{10} (0.001243) + 36.6 \approx 46 \text{dB}$$

The conclusion was that an LNA was needed in the front-end in the wakeup block. Of course this will depend on the range between the transmitter and the receiver, and at this stage a maximum distance of 2 meters was set. [4], [6], [10], [11], [24]

### 3.6 System Design

This chapter describes the final edition, during the thesis work, of the wakeup block in the RF transceiver chip. The description of the system architecture starts with a general description of the whole block and after that each sub block is presented more deeply.

During the thesis work a requirement specification was written for the wakeup block. This document is confidential, but most of its content is to be found here in the report.

#### 3.6.1 General description

The wakeup circuitry is accordingly a RF receiver using the 2.45 GHz ISM-band as the operating frequency and is from now on called wakeup_a in the report. The wakeup_a is working as a wakeup function for a larger circuitry. Its main function is to wait for a specified data packet that is transmitted from a base station on the mentioned frequency. When correct data is received and detected, it should switch on vdd to the other circuitry (the MAC block and the RF block). The wakeup_a circuitry is continuously active during a long time and therefore the total power consumption must be low (<200nA). The power supply to both the digital and analogue parts in the wakeup_a is the unregulated battery voltage (2.1V - 3.6V).

The data packet that is sent from the base station to the implant unit is modulated with OOK and is Manchester encoded. The transmitted data packet is twice as long (in bits counted) as the received packet in the digital part. This is the effect of using the mentioned special decoder block (from now on called wake_dec). This block decodes different patterns consisting of different time variables concerning presence or absence of the RF signal. These patterns and the explanation behind them are described further in the chapter about the wake_dec and in the chapter about the data packet.

The wakeup_a consists of an analogue front-end, which amplifies, detects and decodes the incoming RF signal to a data signal and a clock signal. This part is built up by an LNA (wake_lna), a detector (wake_det) and a decoder (wake_dec). The main block in the wakeup_a is the digital control block, wake_ctrl, which interpret and handles the incoming data, both from the detector/decoder and the MAC block. See Figure 18 for better understanding concerning the architectural structure of the wakeup block.
The incoming RF signal needs to be amplified for detection and therefore the LNA (wake_lna) must be used in the receiver. The low power requirements on the circuitry do not allow the wake_lna to be active all the time. A solution to this is to use a strobe signal that activates the LNA according to a specific duty cycle. The strobe signal is extracted in the digital wake_ctrl block from the incoming clock signal from the application. There is also an opportunity to use the direct strobe signal wu_en, which is also coming from the application. The strobe signal is prolonged if a specific pattern is detected in the incoming signal during the active strobe time. The strobe signal also activates the other blocks in the incoming RF part, including the wake_det, wake_dec and wake_bias. The wake_bias is a block that has a specified output of a bias current for the wake_dec and the wake_lna. A register in wake_ctrl controls this block.

The ibs_in signal is coming from the application and determines if the wakeup_a should be either an implant unit or a base station. If the ibs_in is set high, the wakeup_a is an implant and should work as described. If the ibs_in is set low the circuitry should work as a base station. In the base station mode the wakeup_a switch on vdd, both to the MAC and the RF block at once (the same procedure as when a correct message is received, when working as an implanted unit). When working as a base station the wake_lna, the wake_det, the wake_dec and the wake_bias can be completely switched off (strobe signal is set to low). The wake_ctrl is handling all the communication with the MAC block and it is also controlling the enable signals to the wake_vreg that feeds the MAC and RF block with regulated voltage.

The block wake_vreg has voltage regulators for achieving both digital voltage (1.8V) and analogue voltage (2.1-2.2V) for the other circuitry. This block handles the power
supply to all regulated digital parts and all regulated analogue parts in the complete circuitry.

The signal mac_ready from the MAC is set high when the MAC is ready after it has been powered up. When this signal is received the strobe signal is disabled and the wake_lna, wake_det, wake_dec and wake_bias are switched off. When the MAC then later sets the mac_ready signal low, the wake_ctrl block goes back to initial state and listens to the strobe signal again.

The wake_por block is a power on reset function, which holds the device in reset until the operating conditions are met. The interface_ctrl block only contains control gates for having known values on the signals from the MAC to wake_ctrl, when the MAC is without power. The testio bus is for test purpose and the output is controlled via the testio_ctrl signal. A register in the wake_ctrl block sets this signal.

### 3.6.2 Pin description

<table>
<thead>
<tr>
<th>Pin name</th>
<th>Function</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>vdd</td>
<td>Positive supply voltage</td>
<td>Battery voltage, 2,1V-3,6V</td>
</tr>
<tr>
<td>vss</td>
<td>Negative supply voltage</td>
<td></td>
</tr>
<tr>
<td>rx_245</td>
<td>2.45 GHz RF signal input</td>
<td></td>
</tr>
<tr>
<td>data_inout</td>
<td>Transmit/Receive data to MAC in/out</td>
<td>Bi-directional Bus[7:0]</td>
</tr>
<tr>
<td>addr</td>
<td>Addresses to registers from MAC input</td>
<td>Bus[6:0]</td>
</tr>
<tr>
<td>rw</td>
<td>Read/write command from MAC input</td>
<td></td>
</tr>
<tr>
<td>str</td>
<td>Strobe from MAC input</td>
<td></td>
</tr>
<tr>
<td>mac_ready</td>
<td>Power supply OK from MAC input</td>
<td></td>
</tr>
<tr>
<td>wu_en</td>
<td>direct strobe signal input</td>
<td>from Application</td>
</tr>
<tr>
<td>wu_clk</td>
<td>4-32kHz clock signal input</td>
<td>from Application</td>
</tr>
<tr>
<td>ibs_in</td>
<td>Switch for IU or BS mode input</td>
<td>from Application</td>
</tr>
<tr>
<td>rf_en</td>
<td>enable signal to RF power switches output</td>
<td></td>
</tr>
<tr>
<td>testio</td>
<td>Control bus for test purpose output</td>
<td></td>
</tr>
<tr>
<td>vdda_rf_pa</td>
<td>Power feed rf_pa output</td>
<td>2,1V-2,2V</td>
</tr>
<tr>
<td>vdda_rf_vco</td>
<td>Power feed rf_vco output</td>
<td>2,1V-2,2V</td>
</tr>
<tr>
<td>vdda_rf_gen</td>
<td>Power feed rf_gen output</td>
<td>2,1V - 2,2V (LO, mixer etc.)</td>
</tr>
<tr>
<td>vdda_rf_lna</td>
<td>Power feed rf_lna output</td>
<td>2,1V-2,2V</td>
</tr>
<tr>
<td>vdda_rf_xo</td>
<td>Power feed xo_rf output</td>
<td>2,1V-2,2V</td>
</tr>
<tr>
<td>vdda</td>
<td>Power feed for de-coupling cap output</td>
<td>2,1V-2,2V</td>
</tr>
<tr>
<td>vddd_gen</td>
<td>Power feed general digital output</td>
<td>1,8V</td>
</tr>
</tbody>
</table>

Table 1 Pin description to the wakeup block
3.6.3 The LNA – wake_lna

The Low Noise Amplifier, called wake_lna, is used because the incoming signal is otherwise too weak for detection. The first approach was to do the receiver without an LNA because it would probably cost too much in power to use one. Later, due to field measurements (see chapter Attenuation on page 29) it was found that the signal was too weak. An LNA has too much power consumption when it is active and that was a problem. If a strobe signal is used for activating the LNA according to a specific time schedule the power consumption can be brought down to an acceptable level. When the signal is amplified it is redirected to the detector. The incoming signal is minimum calculated to 400µV and the output is estimated to be at least minimum 12mV from the LNA, which gives a minimum level of 30 dB gain in the wake_lna. This gain is calculated from the equation below (EQ 2).

\[
\text{dB} = 20 \log \left( \frac{V_{\text{OUT}}}{V_{\text{IN}}} \right) \Rightarrow 20 \log \left( \frac{0.012}{0.0004} \right) = 30 \text{dB} \quad (EQ \ 2)
\]

The maximum current consumption of the active wake_lna is estimated to about 200µA and the total current consumption of the wakeup_a is set to maximum 200nA. This results in a strobe duty cycle of 2000 for the LNA, assuming that the total current consumption of the wakeup block is maximum 100nA, when the wake_lna is inactive. The duty cycle is calculated from the equation below (EQ 3). The total start-up time for the LNA is at this moment estimated to approximately 5µs. [6], [31]

\[
P_{\text{current\_consump\_total}} = P_{\text{current\_consump\_rest}} + P_{\text{current\_consump\_wake\_lna}} \Rightarrow
\]

\[
200nA = 100nA + 200µA \cdot \frac{1}{X} \Rightarrow 200nA - 100nA = \frac{200µA}{X} \Rightarrow X = \frac{200µA}{100nA} = 2000 \quad (EQ \ 3)
\]

3.6.4 The detector – wake_det

The incoming signal to this block is the amplified OOK RF signal. Demodulation or detection of OOK is very basic; the detector just needs to detect the presence or absence of the oscillation. Demodulators that do this and convert the incoming signal into recognizable form are most suitable for low power requirements on receivers.

The incoming signal should at first be rectified in the detector. The detector may consist of a rectifying diode and a capacitor. For really small signals as these (below -20dBm), a peak diode detector (also called envelope detector) can give predictable and useful amplitude measurements. The DC component of a diode output is proportional to the square of the AC input voltage (the so-called square law). If RF input voltage is reduced by half, the output will be one quarter as much DC. If 10 times as much RF input is applied, the output will be 100 times as much DC as it was before. An increase of 3dB results in twice as much output voltage. A diode with a lot of capacitance is out of the question in this case because of the low power requirement and low voltage input. A diode with a sharp I vs. V curve will give more output. Schottky diodes are
great, but one drawback with them is that they require DC biasing to get into the really curved part of the I vs. V curve.

When the signal is rectified it is redirected to a comparator that is fed with a reference voltage $V_{\text{ref}}$. The incoming signal should there be compared to this and clipped at the definite threshold voltage and the output from this stage should be a reasonable square wave, which represents the incoming RF signal. [6], [7], [8], [9], [21], [22]

3.6.5 The decoder – wake_dec

After the detector there is a block that decodes the incoming signal according to the length of the incoming bursts. The incoming signal shall be converted to a decoded output. The signal passes through a circuitry (a RC net) that distinguishes two specified patterns from each other, depending on the burst length. The circuit with the RC net can be described with the principle schematic below in Figure 19. The circuit is in the report referred to as the decoder or the wake_dec, as it was called during the requirement specification phase.

![Figure 19 Schematic over the decoder – the wake_dec](image)

The function of the block is that if the incoming signal consists of a short (S) ‘0’ followed by a short (S) ’1’ the output should be a ’0’. The output will be synchronized with the negative flank on the incoming data signal, used as the clock signal into the digital part. On the other hand, if the input is a long (L) ‘0’ followed by a long (L) ‘1’ the output should be a ‘1’ (also synchronized with the negative flank on the incoming data signal). Figure 20 describes the function of the signal path in the decoder.

![Figure 20 Explanation of the decoding of the signal in the decoder](image)
The incoming signal to this block is accordingly used as the clock for the digital block. Hence, the incoming signal to this block is the clock signal to this block’s output when these two flows into the digital block. The RC net in the wake_dec block has a time factor that distinguishes the input, depending on the burst length in time measured.

**Summary of function in the decoder**

Incoming **short** ‘0’ followed by **short** ‘1’ = ‘0’ as digital output

Incoming **long** ‘0’ followed by **long** ‘1’ = ‘1’ as digital output

The RC net is only sensitive for the time between the negative flanks on the incoming signal according to Figure 21. When the time between two negative flanks on the signal is long enough ($t_{\text{long, min}}$) the RC net will be up-loaded and a 1 will be the output. The figure below gives a hint about what is meant by the time constant for the RC net.

![Figure 21 Time requirements of the RC net in the decoder](image)

$t_{\text{short, max}}$ is the maximum time between negative pulse edges for a short (S) signal and is determined as $X_s$.

$t_{\text{long, min}}$ is the minimum time between negative pulse edges for a long (L) signal and is determined as $3X_s$.

**3.6.6 The digital block - wake_ctrl**

The wake_ctrl block is the digital part in the wakeup_a and it has the control of the awakening of the circuitry. The incoming main signals are the Manchester coded RF signal (used as clock signal) and the RF signal decoded, which is the data signal. The origin of these two is the same and the clock signal is the signal as it was detected and the data signal is the decoded data (output from wake_dec). This solution is used, as mentioned before, because of the high requirement on low power consumption and therefore no internal clock or oscillator could be used for the digital logic. The data signal is clocked in with the clock signal on the negative edge of the pulse. To get an overview of the architectural structure in the digital block, see Figure 22 below.
Since the analogue front-end is strobed during a specific time the wake_ctrl block will start looking for a specific pattern in the incoming data signal. The patterns the wake_ctrl block will search for is 0101, 0110, 1001 or 1010. These combinations are repeated in the sent data packet from the base station. This will happen because of the Manchester encoding that is used. When the wake_ctrl block still receives this pattern or the start sequence the strobe signal (ctrl_str) is held high, but if the pattern is lost the ctrl_str signal will turn low. The start sequence is a signal pattern consisting of 11110000. If the wake_ctrl detects this, then it knows that a correct data packet is coming in and of course the signal ctrl_str is kept high during the download of the following specified number of data bits.

When the start sequence is detected, it sets a flag high and the wake-up control block starts to read in the rest of the data packet, which is Manchester encoded. If there is a match on the company code (the first 8 bits or 16 bits Manchester encoded) and the application ID (the following 26 bits or 52 bits Manchester encoded), the two enable signals, vreg_en and osc_en, are set high. The application ID can also be matched with a master ID, further discussed in the chapter about the data packet. When this is done and the wake_vreg powers up the MAC block and the RF block, the MAC block initiate contact with the wake-up block by setting the mac_ready signal high. When this signal is received the wake_ctrl block sets the osc_en signal low.

The osc_en signal starts a RC oscillator in the wake_vreg, which put out the wdog_clk signal (clock signal). This clock signal is used for a watchdog function, which resets the wakeup_a if the mac_ready signal is not set high within a specified time. This clock signal is generated for this purpose only and nothing else in the logic. Since the
RC oscillator producing this signal is only active for a very short time and not often at all, the total power consumption of this is very low.

After receiving the high mac_ready signal, the wake_ctrl block waits for further instructions by using the data bus system with strobe, addressing and read/write signal. The incoming command can be reading or writing of trim-parameters, application ID, company code or channel ID. Download of the trim-parameters to the wakeup block is done in a special mode in the manufacturing line. There is also a reload mode where the trim parameters can be downloaded again. Upload of the channel ID, application ID and the trim parameters are done each time a correct incoming wakeup message is detected on the 2.45 GHz signal and the MAC is fed with power. The MAC sends a request for these bits by addressing the correct register holding the specific value. The block is holding several trim parameters in different registers for the whole chip and these are also available by addressing the correct register. Some registers are feeding other blocks directly with trim parameters by having hard connections between them, for example the bias_vect to the bias block.

The ibs_in pin sets the wakeup_a in either base station or implant mode. If the pin is set low the circuitry is a base station and the wake_ctrl block powers up the MAC and the RF block directly (same procedure as when being an implanted unit and a correct wakeup message is received). The por_reset is the initial reset that sets the flip-flops and registers to initial states in this block.

The block clk_div handles the strobing procedure. It puts out a strobe signal for the wake_lna and the wake_det according to a specified duty cycle. This clock divider uses the incoming clock from the application for knowing when to strobe and for how long time. The strobe signal can also be directly inserted in this block by setting the input signal wu_en high. The strobe signal is also prolonged if a correct pattern is detected in the incoming signal. For more information and description of the wake_ctrl block see chapter Digital Implementation on page 39.

3.7 The strobing time

The solution with the special decoder, wake_dec, and the strobing procedure with the total analogue front-end, needed a time window for the strobe signal. Since there was already calculated that the minimum duty cycle for the LNA was 2000 (see chapter The LNA – wake_lna on page 33), another calculation was needed to determine the minimum strobing window.

The total time for detecting one of the specified pattern (described in the chapter The digital block - wake_ctrl on page 35) is maximum 8Xs, since a pattern consists of 2 SL or LS packets. To this, the time for a sent 1 (a long signal, L) is added. This is done because the minimum amount of bits that can be received during this time is 5 bits (0 or 1’s). Out of these bits it can then be decided whether it contains one of the specified patterns or not. The total time for strobing the wake_lna will then be 11Xs (8Xs+3Xs).
The minimum time for short 1 and short 0 (0,5Xs, seen in Figure 17 on page 29) is decided by the comparator in the detector stage. During the report was written there were two different solutions of the detector stage uprising. They are called solution A and solution B. The digital implementation done during the thesis work was designed for solution A, because solution B was considered more risky at that time.

### 3.7.1 Solution A

The time 0,5X was in this solution set to 10µs, or X=20µs. This result in a total time of 11*20µs = 220µs for the analogue front-end to be strobed each time. 30µs was added to the total minimum strobe time of 220µs, which result in 250µs as the needed time for detecting one of the specified patterns. Finally to decide the time between two high strobe signals, the value 250µs was multiplied by 2000, which gave 0,5s.

To sum up, the strobe has to be high every 0,5s for 250µs, which is pretty fair. The maximum time for detection will at this moment be 0,5s + time for detecting the message, which is the maximum time for sending 2 messages, 2*(50*4*20µs) = 8ms. The maximum time will then totally be 0,5s + 0,008s = 0,508s for wakeup of the circuitry.

### 3.7.2 Solution B

In this solution the time 0,5X was set to 4µs, giving X=8µs. This result in a total time of 11*8µs = 88µs for the analogue front-end to be strobed each time. 30µs is added to the total minimum strobe time 88µs, which result in 118µs as the needed time for detecting the specified pattern. Finally the time between two high strobe signals were calculated by multiplying the value 118µs with 2000, which gave 0,236s.

To sum up, the strobe has to be high every 0,236s for 118µs, which is very good. The maximum time for detection will in this solution be 0,236s + time for detecting the message, which is the maximum time for sending 2 messages, 2*(50*4*8µs) = 3,2ms. The maximum time will then totally be 0,236s+0,0032s = 0,239s for wakeup of the circuitry.
The thesis included a task that was to implement and design the digital wake_ctrl block, which is described in the chapter about the system architecture. The implementation was done in the Hardware Description Language, VHDL. VHDL is an abbreviation for Very high-speed integrated circuits Hardware Description Language, and is one of the two main HDL-languages in digital design. It is a big and general hardware-description language, which gives several opportunities to describe the same behavior with different language-designs.

The other main language is Verilog and it is a competing hardware design language to VHDL. It is standardized and developed by an American company that was bought by another company named Cadence. It was created before VHDL and thus is more established, foremost in the USA. Verilog was used in the thesis work for writing the testbenches, when testing the design.

The source code, written in VHDL was synthesized using Design Vision from Synopsys. The code was synthesized against the component library for the process 0,18µm from TSMC, which is a mixed signal and RF process. The netlist from the synthesizing was stored as a Verilog file and simulated together with the testbenches using the simulator Verilog-XL. The results from the simulations were examined and reviewed in the program Simvision from Cadence.

The following chapters describe the outcome of the digital implementation done during the design phase of the thesis work. The top-level is first presented and after that each sub block is described more in detail. The source code to each block is found in Appendix B, and the testbench is found in Appendix C.
4.1 Top level – wake_ctrl_a

Figure 23 Top level of the synthesized implementation

Figure 23 presents the top level of the implemented digital design. The figure is an extract from Design Vision, which were where the implementation was synthesized. The design, called wake_ctrl_a, was divided into four main blocks. A special wakeup_package was also created for having all constants in the design, predefined at the same place. This solution has the benefit that it is easy to change a constant, used often in the design (without any difficulties).

The block, start_check_a, search for the specific pattern 0101, 0110, 1001 or 1010 in the incoming data signal, wake_data. If this pattern is detected, the strobe time is prolonged until the pattern is lost or the whole packet is received. The start_check_a block also looks for the pattern 1111 followed by 0000, which is the actual start sequence of the data packet. When this pattern is detected, an enable signal is set to the block wakein_decode_a, which at that moment starts to read in the following incoming data bits.

The Manchester code is checked all the time during the download of the rest of the data to a shift register in the wakein_decode_a block. When the correct amount of bits is received and decoded (from the Manchester code), the shift register is checked against the company code (or spec_compcode) and the application ID (or master ID). If there is a match on these bits then it is assumed that a correct message was received and the signals, wake_vreg and osc_en are set high. At the same time the channel ID is
put into a local register. This register has an output to the block para_data_inout_a, which handles the communication with the MAC block.

When the MAC is powered up and running it sets the signal mac_ready high, which the block para_data_inout_a receives. This signal disables the strobe signal ctrl_str and the analogue front-end of the wakeup circuitry is powered down. After this the para_data_inout_a handles the communication with the MAC by using the bus system consisting of the str (strobe), rw (read='1', write='0'), addr (address bus) and data_inout (data bus). The functions are either read from or write to a register. When the para_data_inout_a block receives the low mac_ready signal, it goes back to initial state and the strobe signal, ctrl_str, is again enabled.

<table>
<thead>
<tr>
<th>Category : Name</th>
<th>To/From</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input : wake_sig</td>
<td>wake_det</td>
<td>raw RF signal used as clock</td>
</tr>
<tr>
<td>Input : wake_data</td>
<td>wake_dec</td>
<td>decoded RF used as data</td>
</tr>
<tr>
<td>Input : addr</td>
<td>MAC</td>
<td>bus [6:0] for addressing correct register</td>
</tr>
<tr>
<td>Input : str</td>
<td>MAC</td>
<td>strobe</td>
</tr>
<tr>
<td>Input : rw</td>
<td>MAC</td>
<td>read/write</td>
</tr>
<tr>
<td>Input : mac_ready</td>
<td>MAC</td>
<td>signal for OK or going to sleep mode</td>
</tr>
<tr>
<td>Input : por_reset</td>
<td>wake_por</td>
<td>power on reset signal</td>
</tr>
<tr>
<td>Input : wu_clk</td>
<td>pad</td>
<td>4-32kHz clock from application</td>
</tr>
<tr>
<td>Input : ibs_in</td>
<td>pad</td>
<td>implant/base selection from application</td>
</tr>
<tr>
<td>Input : wu_en</td>
<td>pad</td>
<td>wakeup enable from application</td>
</tr>
<tr>
<td>Input : wdog_clk</td>
<td>wake_vreg</td>
<td>watchdog clock from RC osc in vreg</td>
</tr>
<tr>
<td>Input/Output : data_inout</td>
<td>MAC</td>
<td>bi-directional bus [7:0] for data</td>
</tr>
<tr>
<td>Output : tr1</td>
<td>wake_lna</td>
<td>trimming flag for LNA</td>
</tr>
<tr>
<td>Output : tr2</td>
<td>wake_lna</td>
<td>trimming flag for LNA</td>
</tr>
<tr>
<td>Output : vreg_en</td>
<td>wake_vreg</td>
<td>enable signal to vreg for VDD</td>
</tr>
<tr>
<td>Output : osc_en</td>
<td>wake_vreg</td>
<td>enable signal to RC-osc for watchdog</td>
</tr>
<tr>
<td>Output : ctrl_str</td>
<td>wake_lna, wake_det, wake_dec, wake_bias</td>
<td>strobe signal</td>
</tr>
<tr>
<td>Output : testio_ctrl</td>
<td>wake_lna, wake_det, wake_dec, wake_bias</td>
<td>bus [7:0] for control of test bus</td>
</tr>
<tr>
<td>Output : bias_vect</td>
<td>wake_bias</td>
<td>bus [7:0] to wake_bias</td>
</tr>
<tr>
<td>Output : sens_vect</td>
<td>wake_det</td>
<td>bus [7:0] to wake_det</td>
</tr>
<tr>
<td>Output : wu_tune</td>
<td>match net</td>
<td>bus [7:0] to antenna matching net</td>
</tr>
</tbody>
</table>

**Table 2 I/O’s for wake_ctrl_a**
4.1.1 Sub block start_check_a

This block handles the first contact with the incoming RF signal. It reads in the bits into a shift register (8 bits long) and then compares the last 4 read in bits with predefined patterns. These patterns are 0101, 0110, 100, 1010 and they initiate that it is a RF package for the wakeup block. These can be found in a correct message for wakeup, because the defined data packet is Manchester encoded. As long as the start_check_a block finds these patterns, the strobe signal is prolonged and the data will continue to be read in. If the receiver has detected a correct message, the start sequence soon appears in the message, since the message is transmitted several times in a row from the base station. The start sequence the block is searching for is 1111 followed by 0000, which is a unique code as mentioned earlier in the report.

When the data starts to be read in to the shift register a counter is started as well. If this counter reaches the number 5, it is assumed that there is no wakeup message and the strobe is not prolonged. If the block finds the pattern the counter is reset. If the beginning of the start sequence (1111) is found the counter is also reset, but it allows only four more bits to be read in and these are then compared to 0000. If there is a match on these bits, the start sequence is found and the enable signal to the wakein_decode_a is set high.

4.1.2 Sub block wakein_decode_a

This block is enabled by the enable signal from the start_check_a block. This signal is set high when the correct start sequence is detected. This block starts to read in the data bits into a shift register, as they arrive. The number of data bits is in the design set to 92 and therefore a counter counts every bit and stops when this number of bits is reached. The shift register is then compared to the company code and the application ID. If there is a match on these bits, the vreg_en and osc_en signals are set high. These signals enable the regulators respectively the RC oscillator in the wake_vreg block.

The incoming data is decoded from the Manchester encoding before it is put into the shift register. At the same time it checks automatically that the Manchester encoding is OK. If the Manchester encoding is not correct, the circuitry is reset and goes back to the initial state.

After detecting the correct wakeup signal, the wakein_decode_a block waits for the mac_ready signal to turn high from the MAC block. If this signal is received, the osc_en signal is turned low and the digital block is ready for communication via the bus system. The para_data_inout_a block handles this communication. When the osc_en signal turned high, it activated an RC oscillator in the wake_vreg block. This RC oscillator extracts a clock signal, which enables a counter that starts to count to a specific number. This is a so-called watch dog function and is implemented as a backup, if the high mac_ready signal never arrives to the block from the MAC. If the watch dog signal from the wake_vreg turns high the digital block is reset and goes back to the initial state.
4.1.3 Sub block para_data_inout_a

The para_data_inout_a block handles the communication on the bus system. A bi-
directional data bus (data_inout), an address bus (addr), a read/write signal (rw) and a
strobe signal (str) builds up the bus system. The command can be either reading or
writing to different registers in this block. The address is set to the bus system together
with the read (=1) or write (=0) signal (if the command is write to a register the data is
also put out to the data_inout bus). Shortly after this the strobe signal is set high and if
the address is correct for a register in this block it performs what the rw signals wants.

For example, if the MAC wants to write in a new value to a register it puts out the
correct address on the address bus and the data is put out to the data bus and the
read/write signal is set to 0. Shortly after this, the MAC sets the strobe signal high and this block does the correct action at this moment. When the MAC assumes that the
action is done, it sets the strobe signal low and a new reading or writing of registers
can be initiated.

Registers in the digital block

The sub block para_data_inout_a contains all the registers in the wakeup_a block. These registers hold trim parameters, company code, application ID, and channel ID
for example. Some trimming procedures are done in the RF block and the registers
here in the wake_ctrl_a block are the memory for the chip. The wakeup_a is the only
block in the whole chip that is fed with vdd continuously. The table below summarizes
the content of the register bank. The full address length is 7 bits, of which the two
MSB’s point to the wakeup_a block. These two bits (11) have been excluded in this
table.

<table>
<thead>
<tr>
<th>Local address, Binary (dec)</th>
<th>Bits, 7=MSB, 0=LSB</th>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00001 (1)</td>
<td>[7:0]</td>
<td>R/W</td>
<td>r00001_appid1</td>
<td>bit 1-8 in application ID</td>
</tr>
<tr>
<td>00010 (2)</td>
<td>[7:0]</td>
<td>R/W</td>
<td>r00010_appid2</td>
<td>bit 9-16 in application ID</td>
</tr>
<tr>
<td>00011 (3)</td>
<td>[7:0]</td>
<td>R/W</td>
<td>r00011_appid3</td>
<td>bit 17-24 in application ID</td>
</tr>
<tr>
<td>00100 (4)</td>
<td>[7:6]</td>
<td>R/W</td>
<td>r00100_appid4</td>
<td>bit 25-26 in application ID</td>
</tr>
<tr>
<td>00101 (5)</td>
<td>[7:0]</td>
<td>R/W</td>
<td>ant_tune</td>
<td>Ant. matching network setting</td>
</tr>
<tr>
<td>00110 (6)</td>
<td>[7:0]</td>
<td>R/W</td>
<td>tx_power_lev</td>
<td>TX_PA output power level setting</td>
</tr>
<tr>
<td>00111 (7)</td>
<td>[7:0]</td>
<td>R/W</td>
<td>lna_gain_control</td>
<td>Gain setting of LNA</td>
</tr>
<tr>
<td>01000 (8)</td>
<td>[7:0]</td>
<td>R/W</td>
<td>cco_current_lev</td>
<td>TX_IF_CCO current setting</td>
</tr>
<tr>
<td>01001 (9)</td>
<td>[7:3]</td>
<td>R/W</td>
<td>tx_freq_sep</td>
<td>Frequency separation setting</td>
</tr>
<tr>
<td></td>
<td>[2]</td>
<td>R/W</td>
<td>storage_para</td>
<td>wakeup storage parameter</td>
</tr>
<tr>
<td></td>
<td>[1]</td>
<td>R/W</td>
<td>tr1</td>
<td>frequency trimming flag</td>
</tr>
<tr>
<td></td>
<td>[0]</td>
<td>R/W</td>
<td>tr2</td>
<td>frequency trimming flag</td>
</tr>
<tr>
<td>01010 (10)</td>
<td>[7:0]</td>
<td>R/W</td>
<td>rx_det_filter_tune</td>
<td>RX IF detector filter setting</td>
</tr>
</tbody>
</table>
Table 3 Registers in the digital block

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>Read/Write</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01011</td>
<td>[7:0]</td>
<td>R/W</td>
<td>rx_if_filter_tune RX_IF_FILTER setting</td>
</tr>
<tr>
<td>01100</td>
<td>[7:0]</td>
<td>R/W</td>
<td>tx_if_filter_tune TX_IF_FILTER setting</td>
</tr>
<tr>
<td>01101</td>
<td>[7:0]</td>
<td>R/W</td>
<td>decision_level RX ADC decision level setting</td>
</tr>
<tr>
<td>01110</td>
<td>[7:0]</td>
<td>R/W</td>
<td>rssi_ref RSSI reference value</td>
</tr>
<tr>
<td>01111</td>
<td>[7:0]</td>
<td>R/W</td>
<td>xo_cap XO setting</td>
</tr>
<tr>
<td>10000</td>
<td>[7:0]</td>
<td>R/W</td>
<td>ref_r Reference resistor setting</td>
</tr>
<tr>
<td>10001</td>
<td>[7:0]</td>
<td>R/W</td>
<td>r10001_compcod specific company code</td>
</tr>
<tr>
<td>10010</td>
<td>[7:0]</td>
<td>R/W</td>
<td>r10010_bias trim parameters for wake_bias</td>
</tr>
<tr>
<td>10011</td>
<td>[7:0]</td>
<td>R/W</td>
<td>r10011_testio control signals for test</td>
</tr>
<tr>
<td>10100</td>
<td>[7:0]</td>
<td>R/W</td>
<td>r10100_sens trim parameters for wake_det</td>
</tr>
<tr>
<td>10101</td>
<td>[7:0]</td>
<td>R/W</td>
<td>r10101_channel channel selection</td>
</tr>
<tr>
<td>10110</td>
<td>[7:4]</td>
<td>R/W</td>
<td>r10110_band band selection</td>
</tr>
<tr>
<td>10111</td>
<td>[7:0]</td>
<td>R/W</td>
<td>wdog10111_xo count time for XO watchdog</td>
</tr>
<tr>
<td>11000</td>
<td>[7:0]</td>
<td>R/W</td>
<td>wdog11000_main count time for main watchdog</td>
</tr>
<tr>
<td>11001</td>
<td>[7:0]</td>
<td>R/W</td>
<td>wdog11001_txrx count time for TX_RX watchdog</td>
</tr>
<tr>
<td>11010</td>
<td>[7:0]</td>
<td>R/W</td>
<td>r11010_wu_tune trim parameter for matchnet</td>
</tr>
</tbody>
</table>

4.1.4 Sub block clk_div_a

This block handles the extraction from the incoming clock signal to a strobe signal for the front-end. In the beginning there was no clock available at all. Later, when some measurements were done with the detector without an LNA, it was found that the signal was too weak for detection for medical implantable applications. The result was to ask for a clock signal from the application. This clock signal is in the implementation set to 32 kHz and it is divided with a manually designed solution in Cadence. This clock divider was chosen because it has lower power consumption compared to a standard divider made in VHDL. The clock divider is found in appendix B under the name strobe_clk2_a.

Two clock signals are extracted from strobe_clk2_a and put into another design made in VHDL called strobe_clk_a. The extracted clock signals are a 4 kHz signal and a 2 Hz signal. These two are then used for making the strobe signal turn high every 0.5s for 250µs. This strobe window is calculated from the given value of the sent message and is depending on the detector in the front-end and the RC net in the decoder block. For more information about this see chapter The strobing time on page 37.
There is also an opportunity to use the signal $wu_{\text{en}}$ for setting the strobe signal. This signal is used if an application wants to calculate the strobe window itself instead of sending a clock signal to the wakeup.

### 4.2 Simulations

When the RTL code was synthesized a netlist was created. The netlist was stored as a Verilog file and simulated for verification together with a testbench by using the simulator Verilog-XL. The results from the simulations were examined and reviewed in the program Simvision from Cadence. The simulation results are shown in this chapter in the extracts from the waveform windows from Simvision.

#### 4.2.1 Top simulation

As seen in Figure 24 the simulation is behaving as wanted. In the beginning everything is reset by the signal $por_{\text{reset}}$. Short after that the incoming data signal, $wake_{\text{data}}$ is started. The $start_{\text{check}}$ block starts to search for one of the four patterns in the incoming signal $wake_{\text{data}}$.

![Figure 24 Top simulation with correct testbench](image)

When the start sequence is detected (approximately at 2.2µs) the enable signal for the $wakein_{\text{decode}}$ block is set high. After this the following 92 bits are read in and decoded to company code, application ID and channel ID. If there is a match on the company code and the application ID, the channel ID is set as an output to the $para_{\text{data}}_{\text{inout}}$ block (approximately at 6.6µs). At the same time the $vreg_{\text{en}}$ and $osc_{\text{en}}$ signals are set high as seen. The $osc_{\text{en}}$ signal is then set low when the incoming signal $mac_{\text{ready}}$ is set high.
Shortly after this (approximately at 8.3 µs) the communication with the MAC on the bus system is started. First the write-to-registers are shown and shortly after that (approximately at 12.6 µs) the read-from-registers are shown. Shortly after 16.4 µs the signal mac_ready from the MAC is set low and the behavior of this is that the whole chip shall return to the initial state, without resetting the registers. The vreg_en signal is set low and so on. The block now again listens to the input signal wake_data and starts to decode this again.

In the second wakeup message (starting at approximately 18.6 µs), the company code is the spec_compcode and the application ID is the appid. These were downloaded to the specific registers during the communication over the bus system earlier (started approximately at 8.4 µs and 12.2 µs). Shortly after 24 µs, the detecting of this message is shown working OK.

4.2.2 Simulation close-ups

For verification of the design, some waveform windows showing close-ups from the simulation is pasted below. These are shown in this chapter for better understanding of a few functions. Some simulations are done with a few changes in the testbench. All changes in the testbench (see Appendix C) are marked with comments and they should be easy to find.

**Figure 25 Write-to-register function via the bus system**

As seen in Figure 25 each register gets its value when the strobe turns high and the correct address is on the address bus.

**Figure 26 Read-from-register function via the bus system**
As seen in Figure 26, data from correct register is put out to the data_inout bus when the strobe is high and correct address is on the address bus. To see that correct address is addressing correct register see Figure 25 above, where each register gets its value.

Figure 27 Decoding of start sequence and prolonging of strobe signal

As seen in Figure 27 the wake_data is read in to the pattern_vect on the negative clock edge of the wake_sig (used as clock). A counter starts to count and if its counts up to more than 5, the strobe signal is set low (for example at 0.85µs and 1.25µs) and no valid pattern is detected. If an OK pattern is detected the strobe signal ctrl_str is prolonged. If the first four bits (7 down to 4) of the pattern_vect is 1111 (approximately at 2.29µs); the only valid pattern for the next four bits is 0000. If this sequence of pattern is detected, the start sequence is found and the enable signal enb is set high (approximately at 2.48µs). This signal is the enable signal for the wakein_decode block, which reads in the rest of the message.

Figure 28 Decoding of the company code and application ID

In Figure 28 (it shows) that two bits are read in after each other to the vector two_vect. This vector is then compared to 01 or 10 for checking and decoding the Manchester code. At the same time the total amount of bits are counted and when the correct
number, in this case 92 or hex 5C, is reached, the register is checked against the application ID and the company code. The waveform also illustrates (approximately at 7,2µs) that a correct channel ID is sent forward and the signals, osc_en and vreg_en are presented.

**Figure 29 Reset of logic when wrong code is incoming (Manchester code)**

In the waveform in Figure 29, the Manchester coding was interrupted (approximately at 20,9µs). The testbench were prepared for this as seen in Appendix B.

**Figure 30 Reset of logic when wrong code is coming in (application ID)**

In this waveform (Figure 30) the application ID did not match, but the Manchester code was OK. The vreg_en and osc_en signal are never set high (approximately at 24µs) after the receiving of the message. The testbench were prepared for this as seen on page 25 in this document.

**Figure 31 Strobing cycle and the function of the wu_en signal**
To get the extraction from the incoming clock signal the simulation time needed to be longer. In the waveform in Figure 31 the testbench were prepared for this as seen in Appendix C in this report.

Figure 32 Function of the ibs_in signal

When the ibs_in signal is set low (approximately at 2µs and 20,8µs in Figure 32) the function shall be the same as when a correct message is detected. Shortly after 2µs the watchdog function is tested (used if the mac_ready signal never turns high) and as seen the logic is reset and goes back to initial state, when the ibs_in is set high and the watch_dog is set low again. At 20,8µs the mac_ready signal turns high as expected and then soon after that it turns low again and everything is reset and goes back to initial state. The testbench were prepared for this as seen on page 17 and 26 in this document.
4.3 Design process

During the digital implementation phase a predefined design process was followed. Figure 33 describes a complete design flow for designing digital blocks. The input to the digital design process was the requirement specification created earlier in the project.

![Digital block design process](image)

Figure 33 Digital block design process

4.3.1 RTL-coding

The actual design process starts with the RTL-coding (Register Transfer Level), which in this project was done in VHDL (in Verilog for the testbench). RTL-coding means capture of the design information in a format ready for direct input to a logic synthesis
Digital Implementation

tool. This capture may go through one or more stages, for example schematic capture of connectivity between sub-blocks followed by automatic generation of structural code followed by addition of behavioral code. As well as design information, a testbench for verification of the RTL code and synthesis scripts are also produced during this stage.

Synthesis scripts should contain realistic constraints and commands for the synthesis tool and the RTL code should be mapped-to-gates to prove that the scripts work efficiently, and to provide an area estimate. The design report document should also be started during this process and should contain enough information to enable a colleague to understand both the RTL code and the synthesis scripts.

The input to this stage is the requirement specification and logic library in formats suitable for synthesis and simulation. The output from this stage should be the source RTL code, testbench and synthesis scripts and constraints.

4.3.2 RTL simulation

This stage is for the verification of the RTL code, to prove that it conforms to the requirement specification. Proof may come through study of the simulation output, through comparison of the simulation output with a set of known good results or through generation of files that can be read into other tools for analysis. Comparison of simulation output with known good results might be achieved in the testbench in an automatic fashion.

The input to this stage is the actual RTL Code, the testbench and behavioral models of any analogue blocks required for co-simulation with the digital block. Sufficient proof needs to be provided to convince an RTL review, so that the block conforms to the specification. This could take the form of printouts, log files and/or a demonstration using a workstation.

4.3.3 Synthesis

Synthesis is the process of compiling the RTL Code to logic expressions, optimizing the logic and mapping the result onto logic gates in a specified technology library. The input to this stage is the RTL code, synthesis set-up file, synthesis scripts, synthesis constraints and layout data.

The synthesis set-up file contains information, which is general to the project, such as the technology library to be used. The command scripts directs the synthesis tool, such as telling it what files to analyze, what compile options to use, to what log file to direct the output etc. The constraints file specifies clock waveforms and environmental constraint, such as input delay and drive capability of each input and output delay and load of each output. Layout data is needed for physical synthesis where the block aspect ratio and pin positions need to be known for placement of gates.
Chapter 4

The outputs from this stage are gate-level netlist (in Verilog), test protocol file, data base file, log file, placement information and parasitic data, an updated design report. The netlist in Verilog, placement information (.pdef file) and parasitic data (.sdf file) are used by the layout tool; the parasitic data can be used for constraint-driven routing. The test protocol file is used for generating scan test patterns.

4.3.4 Pre-layout verification

Pre-layout verification confirms that the design conforms to specification. Once a gate-level netlist exists there is a direct correspondence between the logic cells in the netlist and the library cells placed on the silicon; this means more types of analysis are now possible, e.g. static timing analysis (STA). In addition, formal verification tools can be used to prove mathematically that the gate-level description produced by the synthesis tool has the same functionality as the RTL code.

The inputs to this stage are the gate-level netlist in Verilog, STA script, synthesis constraints and scripts for other tools that carry out formal verification or power analysis, a testbench in Verilog. The output of this process is a proof of conformance, as in the RTL-simulation stage. In practice this means sufficient proof to convince an audience of people that the block conforms to specification.

4.3.5 Layout

Block layout takes a flat gate-level netlist containing connectivity information, target area and shape of required layout, pin positions, placement data and timing constraints and performs automated cell placement and routing to produce a fully placed and routed DEF netlist. Also produced are an RC parasitics report for timing verification, an abstract in LEF format for top-level routing and a stream file for physical verification. If physical synthesis is used then the placement of cells will already have been done during the synthesis process. The layout process will also insert a power frame around the block and a power mesh over it to which individual cells are connected.

The inputs to this stage are the Verilog netlist, required area of layout, required shape of layout, required pin positions, placement data (if physical synthesis is used), timing constraints (if physical synthesis is not used). The output is a fully placed and routed DEF netlist, file of RC parasitics, abstract in LEF format for top-level routing, stream file of layout data for physical verification.

4.3.6 Post layout verification

Post-layout verification provides the final confirmation that the block conforms to specification. Once a placed and routed gate-level netlist exists accurate information on RC parasitics is available enabling accurate Static Timing Analysis (STA). Physical verification (i.e. DRC, ERC and LVS) checks are also run on the layout to show that the block can be manufactured and that the layout has the same connectivity as the
original netlist. A formal verification tool can be used to confirm that the netlist extracted from the layout is the same as that produced by the synthesis process.

The input to this stage is the gate-level netlist in Verilog, STA script, synthesis constraints, scripts for other tools that carry out formal verification or power analysis, a testbench in Verilog. The output of this process is a proof of conformance to the specification as in the other verification stages.
5 Results

The final data packet, shown on page 27 in the report, is one of the outcomes of the thesis work. The packet was developed continuously during the progress of the project. Once the data packet was defined the incoming RF stage could be investigated. A requirement specification for the wakeup block was written during the system analysis part. This document is confidential but most of the information from this is found in this report.

The final proposal to a complete system design for the wakeup block in the RF transceiver is shown on page 31. The front-end consists of an LNA, a simple detector and a special decoder. Since there was an intention not to have a clock signal available in the digital part, (because of this) self-clocking methods on the incoming RF signal had to be investigated. The solution to this was the special decoder shown on page 34. The decoder distinguishes the incoming RF signal concerning the burst lengths in time. The decoder consists of an RC net that is uploaded and then has an output of 1, if the burst length is long enough and vice versa.

After field measurements according to attenuation in human tissues (see page 29 for more information), it was decided to use an LNA in the front-end. This LNA could not be active all the time, because of the requirements on low power consumption (<200nA) in the wakeup block. The solution to this was to use a strobe signal for the complete front-end, which activates it. This strobe signal was extracted in the digital logic. The strobe signal has a specific duty cycle, depending on the time factors in the detector and in the decoder in the front-end. The strobing time is at this moment high for approximately 250µs every 0.5s (for more information see chapter The strobing time on page 37). This is the strobe signal implemented in the digital solution also done in this thesis work.
The implementation of the digital block is made in VHDL (source code) and Verilog (testbenches). The source code was synthesized against the component library for the process 0.18µm from TSMC, which is a mixed signal and RF process. The netlist from the synthesizing was stored as a Verilog file and simulated together with the testbenches using the simulator Verilog-XL. The results from the simulations were examined and reviewed in the program Simvision from Cadence. The result was then verified during a pre-layout review together with colleagues at Zarlink Semiconductor AB. During the implementation phase a Design report was written continuously and then used for the pre-layout review. Extracts (source code and testbench) from this document can be found as appendixes to this report.

5.1 Future improvements

Some minor changes in the wakeup block will probably come up. Hopefully the main structure will stay as it is but there will probably be minor changes on some of the different sub blocks.

New functions to add to the digital design are very likely to arise. These should be easy to implement in the design. If there for example will be any changes to the data packet, all constants in the source code are easy to change because of the wakeup_package used in the design. The structure of the RTL-code is hierarchical and it is therefore easy to implement more blocks if necessary.

At this moment the digital design process has reached the layout stage and the next task is to do post-layout simulations with updated timing files and parasitic extractions from the layout stage.
6 References


   http://www.ite.mh.se/~bornor/ee/dB.pdf

   http://www.mycal.net/old/projects/digtal/encode.html


    http://www.tuanistechnology.com/education/calculators/fsl.htm

    http://www.quickbuilder.co.uk/qb/articles/
http://www.erg.abdn.ac.uk/users/gorry/course/phy-pages/man.html

http://www.erg.abdn.ac.uk/users/gorry/course/phy-pages/dpll.html

[15] Packet Error Rate Measurements Using the PRISM® Chip Set (2003-02-12)
http://www.intersil.com/data/AN/AN9623.pdf

[16] Error Detection (2003-02-12)
http://www.eecs.tulane.edu/courses/cpen472/slides/ErrorsDetection.pdf


http://www.ciphersbyritter.com/ARTS/CRCMYST.HTM

http://www.easics.com/webtools/crctool


[22] Amplifying DC Diode Detector (2002-12-02)
http://engphys.mcmaster.ca/~elmer101/sqlaw/detcircu.html

http://www.math.chalmers.se/~jub/vetenskap01.pdf

[24] Absorption of RF Radiation (2002-12-12)

www.pts.se

www.etsi.org

[27] Digital modulation, ASK, FSK and PSK (2002-12-05)
http://www.cs.ucl.ac.uk/staff/S.Bhatti/D51-notes/node12.html#figbpsk


[29] RF Avenue (2002-12-05)
http://www.newwaveinstruments.com/resources/rf_microwave_resources/sections/digital_modulation_tutorial_theory_technique.htm

[30] PSK Demodulation (2002-12-05)
[31] Pulse Timing (2003-02-02)

   http://www.tfe.umu.se/courses/dator/Digital1/Paj-Sors_ht01/Litt/vhdl_kom.pdf

[33] VHDL Tutorial: Learn by Example (2003-02-07)
   http://www.cs.ucr.edu/content/esd/labs/tutorial/
# Appendix A  Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC</td>
<td>Alternating Current</td>
</tr>
<tr>
<td>ASK</td>
<td>Amplitude Shift Keying</td>
</tr>
<tr>
<td>BS</td>
<td>Base station</td>
</tr>
<tr>
<td>CDMA</td>
<td>Code Division Multiple Access</td>
</tr>
<tr>
<td>CMOS</td>
<td>Complementary Metal Oxide Semiconductor</td>
</tr>
<tr>
<td>CRC</td>
<td>Cyclic Redundancy Check</td>
</tr>
<tr>
<td>CT2</td>
<td>Cordless Telephone 2</td>
</tr>
<tr>
<td>DC</td>
<td>Direct Current</td>
</tr>
<tr>
<td>DECT</td>
<td>Digital Enhanced Cordless Telephone</td>
</tr>
<tr>
<td>DVB-S</td>
<td>Digital Video Broadcasting – Satellite</td>
</tr>
<tr>
<td>EFTA</td>
<td>European Free Trade Association</td>
</tr>
<tr>
<td>ETSI</td>
<td>European Telecommunications Standards Institute</td>
</tr>
<tr>
<td>FM</td>
<td>Frequency Modulation</td>
</tr>
<tr>
<td>FSK</td>
<td>Frequency Shift Keying</td>
</tr>
<tr>
<td>IF</td>
<td>Intermediate Frequency</td>
</tr>
<tr>
<td>IU</td>
<td>Implanted Unit</td>
</tr>
<tr>
<td>ISM</td>
<td>Industrial, Scientific and Medical</td>
</tr>
<tr>
<td>I/Q</td>
<td>Inphase/Quadrature</td>
</tr>
<tr>
<td>LNA</td>
<td>Low Noise Amplifier</td>
</tr>
<tr>
<td>LO</td>
<td>Local Oscillator</td>
</tr>
<tr>
<td>MAC</td>
<td>Media Access Controller</td>
</tr>
<tr>
<td>MICS</td>
<td>Medical Implant Communication System</td>
</tr>
<tr>
<td>OOK</td>
<td>On/Off Keying</td>
</tr>
<tr>
<td>PA</td>
<td>Power Amplifier</td>
</tr>
<tr>
<td>PCB</td>
<td>Printed Circuit Board</td>
</tr>
<tr>
<td>PDA</td>
<td>Personal Digital Assistant</td>
</tr>
<tr>
<td>PSK</td>
<td>Phase Shift Keying</td>
</tr>
<tr>
<td>RF</td>
<td>Radio Frequency</td>
</tr>
</tbody>
</table>
### Definitions

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dB</td>
<td>Deci-Bell</td>
</tr>
<tr>
<td>dBm</td>
<td>dB relative to 1mW</td>
</tr>
</tbody>
</table>
The design was implemented using VHDL as HDL-language and then synthesized using Design Vision from Synopsys. The netlist was stored as Verilog file and simulated in Verilog XL with testbenches written in Verilog.

A package called wakeup_package was created, which includes all constants used in the other entities. The code for this is presented first and after that the main block called wake_ctrl, which includes the start_check, wakein_decode, para_data_inout and clk_div are presented.

**wakeup_package**

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

PACKAGE wakeup_package IS

CONSTANT companycode: INTEGER := 16; --man. coded, actual vector length divided by 2
CONSTANT applicationid: INTEGER:=52; --man. coded, actual vector length divided by 2
CONSTANT channelsel: INTEGER:=24; --man. coded, actual vector length divided by 2
CONSTANT startsequence: INTEGER:=8; --actual vector length of startsequence (not man. coded)
CONSTANT fullcount: INTEGER:=7; --vector length of counter (min total bits)
CONSTANT addr_bus_width: INTEGER:=7; --address bus width
CONSTANT data_bus_width: INTEGER:=8; --data bus width

--Hardcoded companycode
SUBTYPE comp_vect IS STD_LOGIC_VECTOR((companycode/2)-1 DOWNTO 0);
CONSTANT companycode_vect: comp_vect:="11111111";

--Hardcoded Master ID
SUBTYPE master_vect IS STD_LOGIC_VECTOR((applicationid/2)-1 DOWNTO 0);
CONSTANT Masterid_vect: master_vect:="01010101010101010101010101";

--Hardcoded Master ID & companycode
SUBTYPE compmaster_vect IS STD_LOGIC_VECTOR(((companycode+applicationid)/2)-1 DOWNTO 0);
CONSTANT compandmasterid_vect: compmaster_vect:= Masterid_vect & companycode_vect;

END wakeup_package;
```

**wake_ctrl**

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE work.wakeup_package.ALL;
```
ENTITY wake_ctrl_a IS
PORT( wake_sig: IN STD_LOGIC;
    wake_data: IN STD_LOGIC;
    mac_ready: IN STD_LOGIC;
    wu_CLK: IN STD_LOGIC;
    wu_en: IN STD_LOGIC;
    ibs_in: IN STD_LOGIC;
    str: IN STD_LOGIC;
    por_reset: IN STD_LOGIC;
    watch_dog: IN STD_LOGIC;
    addr: IN STD_LOGIC_VECTOR(addr_bus_width-1 DOWNTO 0);
    data_inout: INOUT STD_LOGIC_VECTOR(data_bus_width-1 DOWNTO 0);
    vreg_en: OUT STD_LOGIC;
    osc_en: OUT STD_LOGIC;
    ctrl_str: OUT STD_LOGIC;
    tr1: OUT STD_LOGIC;
    tr2: OUT STD_LOGIC;
    bias_vect: OUT STD_LOGIC_VECTOR(data_bus_width-1 DOWNTO 0);
    sens_vect: OUT STD_LOGIC_VECTOR(data_bus_width-1 DOWNTO 0);
    testio_ctrl: OUT STD_LOGIC_VECTOR(data_bus_width-1 DOWNTO 0);
    wu_tune: OUT STD_LOGIC_VECTOR(data_bus_width-1 DOWNTO 0)
);
END wake_ctrl_a;
ARCHITECTURE rtl OF wake_ctrl_a IS
SIGNAL strobe_sig, decode_reset_sig, para_reset_sig, reset_sig, enb, wake_out, osc_en_sig,
listen_strobe_sig: STD_LOGIC;
SIGNAL channel: STD_LOGIC_VECTOR((channelsel/2)-1 DOWNTO 0);
SIGNAL appid: STD_LOGIC_VECTOR((applicationid/2)-1 DOWNTO 0);
SIGNAL spec_compcode: STD_LOGIC_VECTOR((companycode/2)-1 DOWNTO 0);
COMPONENT start_check_a
PORT( clk_in_pre: IN STD_LOGIC;
    data_in_pre: IN STD_LOGIC;
    reset_pre: IN STD_LOGIC;
    Out_Enb_pre: OUT STD_LOGIC;
    start_lna_strobe: OUT STD_LOGIC);
END COMPONENT;
COMPONENT wakein_decode_a
PORT( clk_in_code: IN STD_LOGIC;
    data_in_code: IN STD_LOGIC;
    in_enb_code: IN STD_LOGIC;
    wu_ibs_in: IN STD_LOGIC;
    in_mac_ready: IN STD_LOGIC;
    in_por_reset: IN STD_LOGIC;
    in_watch_dog: IN STD_LOGIC;
    appid_vect_in: IN STD_LOGIC_VECTOR((applicationid/2)-1 DOWNTO 0);
    spec_compcode_in: IN STD_LOGIC_VECTOR((companycode/2)-1 DOWNTO 0);
    wrong_detect: OUT STD_LOGIC;
    wake_sig: OUT STD_LOGIC;
    listen_strobe: OUT STD_LOGIC;
    osc_en_out: OUT STD_LOGIC;
    data_out: OUT STD_LOGIC_VECTOR((channelsel/2)-1 DOWNTO 0)
);
END COMPONENT;
COMPONENT para_data_inout_a
PORT( strobe: IN STD_LOGIC;
    para_rw: IN STD_LOGIC;
    para_ibs_in: IN STD_LOGIC;
    channel_in: IN STD_LOGIC_VECTOR((channelsel/2)-1 DOWNTO 0);
    data_in_out: INOUT STD_LOGIC_VECTOR(data_bus_width-1 DOWNTO 0);
    spec_compcode_out: OUT STD_LOGIC_VECTOR((companycode/2)-1 DOWNTO 0);
    wrong_detect: OUT STD_LOGIC;
    para_bias_vect: OUT STD_LOGIC_VECTOR(data_bus_width-1 DOWNTO 0);
    para_atest_vect: OUT STD_LOGIC_VECTOR(data_bus_width-1 DOWNTO 0);
    para_sens_vect: OUT STD_LOGIC_VECTOR(data_bus_width-1 DOWNTO 0);
    para_wu_tune: OUT STD_LOGIC_VECTOR(data_bus_width-1 DOWNTO 0);
    para_reset_out: OUT STD_LOGIC;
    para_tr1: OUT STD_LOGIC;
    para_tr2: OUT STD_LOGIC
);
END COMPONENT;
COMPONENT clk_div_a
PORT( mwu_clk_in: IN STD_LOGIC;
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE work.wakeup_package.ALL;

ENTITY start_check_a IS
  PORT(
    clk_in_pre: IN STD_LOGIC;
    data_in_pre: IN STD_LOGIC;
    Out_Enb_pre: OUT STD_LOGIC;
  );
END COMPONENT;

BEGIN

  vreg_en <= wake_out;
  osc_en <= osc_en_sig;

  G1: start_check_a port map(clk_in_pre => wake_sig,
                              data_in_pre => wake_data,
                              Out_Enb_pre => enb,
                              reset_pre => reset_sig,
                              start_lna_strobe => strobe_sig
                          );

  G2: wakein_decode_a port map(clk_in_code => wake_sig,
                                data_in_code => wake_data,
                                in_enb_code => enb,
                                wake_sig => wake_out,
                                osc_en_out => osc_en_sig,
                                data_out => channel,
                                appid_vect_in => appid,
                                spec_compcode_in => spec_compcode,
                                wrong_detect => decode_reset_sig,
                                wake_ibs_in => ibs_in,
                                in_mac_ready => mac_ready,
                                in_por_reset => por_reset,
                                in_watch_dog => watch_dog,
                                listen_strobe => listen_strobe_sig
                            );

  G3: para_data_inout_a port map(strobe => str,
                                 para_rw => rw,
                                 addr_vect => addr,
                                 data_in_out => data_inout,
                                 channel_in => channel,
                                 appid_vect_out => appid,
                                 spec_compcode_out => spec_compcode,
                                 para_bias_vect => bias_vect,
                                 para_atest_vect => testio_ctrl,
                                 para_sens_vect => sens_vect,
                                 para_por_reset => por_reset,
                                 para_reset_out => para_reset_sig,
                                 para_ibs_in => ibs_in,
                                 para_wu_tune => wu_tune,
                                 para_tr1 => tr1,
                                 para_tr2 => tr2
                        );

  G4: clk_div_a port map(mwu_clk_in => wu_clk,
                        start_strobe => strobe_sig,
                        in_por_reset => por_reset,
                        decode_reset => decode_reset_sig,
                        in_listen_strobe => wake_out,
                        in_para_ibs_reset => para_reset_sig,
                        in_wu_en => wu_en,
                        out_ctrl_str => ctrl_str,
                        reset_out => reset_sig,
                        in_watch_dog => watch_dog
                    );

END rtl;

start_check_a
 Appendix B

```vhdl
reset_pre: IN STD_LOGIC;
start_lna_strobe: OUT STD_LOGIC
);
END start_check_a;
ARCHITECTURE rtl OF start_check_a IS
SIGNAL pattern_vect: STD_LOGIC_VECTOR(startsequence-1 DOWNTO 0);  --8 bits long shiftregister
containing 8 last bits
SIGNAL lna_high, reset_b, first_four_ok, reset: STD_LOGIC;
SUBTYPE count IS INTEGER RANGE 0 TO (startsequence);  -- counter for the correct amount of bits
SIGNAL counter: count;
BEGIN
start_lna_strobe <= lna_high AND lna_high_b;
reset <= reset_pre AND reset_b;

pattern_detect: PROCESS(clk_in_pre, reset_pre)
BEGIN
CASE reset_pre IS
WHEN '1' => IF (FALLING_EDGE(clk_in_pre)) THEN
CASE first_four_ok IS
WHEN '0' => CASE counter IS
WHEN 0 TO 5 => CASE pattern_vect(startsequence-1 DOWNTO startsequence-4) IS
WHEN "0101" => lna_high <='1';
pattern_vect <= data_in_pre & pattern_vect(startsequence-1 DOWNTO 1);
counter<=1;
WHEN "0110" => lna_high <='1';
pattern_vect <= data_in_pre & pattern_vect(startsequence-1 DOWNTO 1);
counter<=1;
WHEN "1001" => lna_high <='1';
pattern_vect <= data_in_pre & pattern_vect(startsequence-1 DOWNTO 1);
counter<=1;
WHEN "1010" => lna_high <='1';
pattern_vect <= data_in_pre & pattern_vect(startsequence-1 DOWNTO 1);
counter<=1;
WHEN "1111" => lna_high <='1';
pattern_vect <= data_in_pre & pattern_vect(startsequence-1 DOWNTO 1);
counter<=1;
first_four_ok <='1';
WHEN OTHERS => pattern_vect <= data_in_pre & pattern_vect(startsequence-1 DOWNTO 1);
counter<=counter+1;
END CASE;
WHEN OTHERS =>
END CASE;
WHEN '1' => CASE counter IS
WHEN 1 TO 3 => pattern_vect <= data_in_pre & pattern_vect(startsequence-1 DOWNTO 1);
counter<=counter+1;
WHEN 4 => CASE pattern_vect(startsequence-1 DOWNTO 0) IS
WHEN "00011111" => lna_high<='1';
WHEN OTHERS => first_four_ok <= '0';
counter<=0;
pattern_vect <= (OTHERS => '0');
lna_high<='0';
END CASE;
END CASE;
END IF;
WHEN OTHERS => pattern_vect <= (OTHERS => '0');
counter <= 0;
first_four_ok <= '0';
lna_high<='0';
END CASE;
END PROCESS pattern_detect;
P3: PROCESS (counter)
BEGIN
CASE counter IS
WHEN 6 => reset_b <= '0';
WHEN OTHERS => reset_b <= '1';
END CASE;
END PROCESS P3;
fast_out: PROCESS (pattern_vect, reset_pre)
BEGIN
CASE reset_pre IS
WHEN '1' => CASE pattern_vect(startsequence-1 DOWNTO 0) IS
WHEN "00011111" => Out_Enb_pre<='1';   --If correct start sequence, enable out
WHEN OTHERS => Out_Enb_pre<='0';   --If NOT correct start sequence, reset out
END CASE;
```

- 66 -
WHEN OTHERS=> Out_Enb_pre<= '0';
END CASE;
END PROCESS fast_out;
END rtl;

wakein_decode_a

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE work.wakeup_package.ALL;

ENTITY wakein_decode_a IS
  PORT( clk_in_code: IN STD_LOGIC;
        data_in_code: IN STD_LOGIC;
        in_enb_code: IN STD_LOGIC;
        wake_ibs_in: IN STD_LOGIC;
        in_mac_ready: IN STD_LOGIC;
        in_por_reset: IN STD_LOGIC;
        in_watch_dog: IN STD_LOGIC;
        spec_compcode_in: IN STD_LOGIC_VECTOR((companycode/2)-1 DOWNTO 0);
        wrong_detect: OUT STD_LOGIC;
        wake_sig: OUT STD_LOGIC;
        listen_strobe: OUT STD_LOGIC;
        osc_en_out: OUT STD_LOGIC;
        data_out: OUT STD_LOGIC_VECTOR((channelsel/2)-1 DOWNTO 0);
        appid_vect_in: IN STD_LOGIC_VECTOR((applicationid/2)-1 DOWNTO 0)
  );
END wakein_decode_a;

ARCHITECTURE rtl OF wakein_decode_a IS
  SIGNAL correct_channel_vect: STD_LOGIC_VECTOR((channelsel/2)-1 DOWNTO 0);
  SIGNAL two_vect: STD_LOGIC_VECTOR(1 DOWNTO 0);
  SIGNAL correct_code_vect: STD_LOGIC_VECTOR(((companycode+applicationid+channelsel)/2)-1 DOWNTO 0);
  SIGNAL man_code, store_data, correct_channel, check_code, reset_int, nada, osc_en_sig, wake_sig_int:
    STD_LOGIC;
  SIGNAL count_code: STD_LOGIC_VECTOR(fullcount DOWNTO 0);
  SIGNAL compandappid_vect, speccompandappid_vect, speccompandmasterid_vect:
    STD_LOGIC_VECTOR(((companycode+applicationid)/2)-1 DOWNTO 0);
  BEGIN
    wrong_detect <= reset_int OR nada;
    wake_sig <= wake_sig_int AND (NOT in_watch_dog);
    data_out <= correct_channel_vect;
    compandappid_vect <= appid_vect_in & companycode_vect;
    speccompandappid_vect <= appid_vect_in & spec_compcode_in;
    speccompandmasterid_vect <= Masterid_vect & spec_compcode_in;

    P1: PROCESS(clk_in_code, in_enb_code, count_code)
    BEGIN
      SUBTYPE minicount IS INTEGER RANGE 0 TO 1;  -- counter for the correct amount of bits
      VARIABLE count_one: minicount;

      BEGIN
        IF (in_enb_code='0') THEN
          count_code <= (OTHERS =>'0');
          count_one :=0;
          man_code <= '0';
        ELSE
          IF (FALLING_EDGE(clk_in_code)) THEN
            IF (count_code <= (companycode+applicationid+channelsel-1)) THEN
              CASE count_one IS
                WHEN 0 => store_data <= data_in_code;
                count_code<=count_code+1;
                count_one:=count_one+1;
                man_code <= '0';
                WHEN OTHERS => two_vect <= data_in_code & store_data;
                count_code:=count_code+1;
                count_one:=0;
                man_code <= '1';
              END CASE;
            ELSE
              count_one:=0;
              man_code <= '0';
            END IF;
          END IF;
        END IF;
      END PROCESS P1;

    P2: PROCESS (clk_in_code, man_code, two_vect, correct_code_vect, in_enb_code)
    BEGIN

CASE man_code IS
  WHEN '1' => IF (RISING_EDGE(clk_in_code)) THEN
      CASE two_vect IS
        WHEN "10" => correct_code_vect <= '0' & correct_code_vect(((companycode+applicationid+channelsel)/2)-1 DOWNTO 1);
        ELSE
          check_code <= '0';
  END IF;
  WHEN "01" => correct_code_vect <= '1' & correct_code_vect(((companycode+applicationid+channelsel)/2)-1 DOWNTO 1);
        IF (count_code = (companycode+applicationid+channelsel)) THEN
          check_code <='1';
        ELSE
          check_code <='0';
  END IF;
  WHEN OTHERS => correct_code_vect <= '0';
      END CASE;
  END IF;
  WHEN OTHERS => IF (in_enb_code='0') THEN
      correct_code_vect <= (OTHERS => '0');
      check_code <= '0';
  END IF;
END CASE;
END PROCESS P2;

P3: PROCESS (wake_ibs_in, in_enb_code, clk_in_code, check_code, correct_code_vect)
BEGIN
  CASE wake_ibs_in IS
    WHEN '1' => CASE in_enb_code IS
      WHEN '1' => IF (check_code='1') THEN
        IF (FALLING_EDGE(clk_in_code)) THEN
          IF (correct_code_vect(((companycode+applicationid)/2)-1 DOWNTO 0) = compandmasterid_vect OR
             correct_code_vect(((companycode+applicationid)/2)-1 DOWNTO 0) = compandappid_vect OR
             correct_code_vect(((companycode+applicationid)/2)-1 DOWNTO 0) = speccompandappid_vect OR
             correct_code_vect(((companycode+applicationid)/2)-1 DOWNTO 0) = speccompandmasterid_vect) THEN
            correct_channel_vect <= correct_code_vect(((companycode+applicationid+channelsel)/2)-1 DOWNTO
             ((companycode+applicationid)/2));
            wake_sig_int <= '1';
            osc_en_sig <= '1';
            listen_strobe <= '0';
          ELSE
            reset_int <= '1';
            END IF;
        END IF;
      ELSE
        END IF;
    WHEN OTHERS => reset_int <= '0';
      END CASE;
    WHEN OTHERS => wakeSig_int <= '1';
      osc_en_sig <= '1';
  END CASE;
END PROCESS P3;

P4: PROCESS (in_enb_code, in_mac_ready)
BEGIN
  CASE in_enb_code IS
    WHEN '0' => nada<='0';
    WHEN OTHERS => IF (FALLING_EDGE(in_mac_ready)) THEN
      nada <='1';
    END IF;
  END CASE;
END PROCESS P4;

P5: PROCESS (in_por_reset, in_mac_ready, osc_en_sig, in_watch_dog)
BEGIN
  IF (in_por_reset='1') THEN
    IF (in_mac_ready='0' AND in_watch_dog='0') THEN
      OSC_en_out <= '1';
      END IF;
    ELSE
      OSC_en_out <= '0';
      END IF;
  ELSE
    OSC_en_out <= '0';
    END IF;
  END PROCESS P5;
END RTL;
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE work.wakeup_package.ALL;

ENTITY para_data_inout_a IS
  PORT( strobe: IN STD_LOGIC;
    para_rw: IN STD_LOGIC;
    addr_vect: IN STD_LOGIC_VECTOR(addr_bus_width-1 DOWNTO 0);
    data_in_out: INOUT STD_LOGIC_VECTOR(data_bus_width-1 DOWNTO 0);
    para_ibs_in: IN STD_LOGIC;
    para_por_reset: IN STD_LOGIC;
    channel_in: IN STD_LOGIC_VECTOR((channelsel/2)-1 DOWNTO 0);
    spec_Comcode_out: OUT STD_LOGIC_VECTOR((companycode/2)-1 DOWNTO 0);
    para_bias_vect: OUT STD_LOGIC_VECTOR(data_bus_width-1 DOWNTO 0);
    para_atest_vect: OUT STD_LOGIC_VECTOR(data_bus_width-1 DOWNTO 0);
    para_sens_vect: OUT STD_LOGIC_VECTOR(data_bus_width-1 DOWNTO 0);
    para_wu_tune: OUT STD_LOGIC_VECTOR(data_bus_width-1 DOWNTO 0);
    para_reset_out: OUT STD_LOGIC;
    para_tr1: OUT STD_LOGIC;
    para_tr2: OUT STD_LOGIC
  );
END para_data_inout_a;

ARCHITECTURE rtl OF para_data_inout_a IS
  SIGNAL data_out_temp, reg_00000, r10001_compcode, r00001_appid1, r00010_appid2, r00011_appid3,
  r00100_appid4, r10101_channel, r10110_band, reg_11111, r10010_bias, r10011_atest, r10100_sens, ant_tune,
  tx_power_lev, lna_gain_control, cco_current_lev, tx_freq_sep, rx_det_filter_tune, rx_if_filter_tune,
  tx_if_filter_tune, decision_level, rssi_ref, xo_cap, ref_r, wdog10111_xo, wdog11000_main, wdog11001_txxr,
  r11010wu_tune: STD_LOGIC_VECTOR(data_bus_width-1 DOWNTO 0);
  SIGNAL right_addr, write, read, nreset: STD_LOGIC;
BEGIN
  write <= (right_addr AND para_rw);
  read <= (right_addr AND NOT para_rw);
  appid_vect_out <= r00001_appid1 & r00010_appid2 & r00011_appid3 & r00100_appid4(data_bus_width-1 DOWNTO
  data_bus_width-2);
  spec_comcode_out <= r10001_compcode;
  data_in_out <= data_out_temp;
  para_bias_vect <= r10010_bias;
  para_atest_vect <= r10011_atest;
  para_sens_vect <= r10100_sens;
  para_wu_tune <= r11010wu_tune;
  para_reset_out <= nreset;
  para_tr1 <= tx_freq_sep(data_bus_width-7);
  para_tr2 <= tx_freq_sep(data_bus_width-8)
P0: PROCESS (addr_vect)
BEGIN
  CASE addr_vect(addr_bus_width-1 DOWNTO addr_bus_width-2) IS
    WHEN "11" => right_addr <= '1';
    WHEN OTHERS => right_addr <= '0';
  END CASE;
END PROCESS P0;

P1: PROCESS (write, strobe, addr_vect, ant_tune, tx_power_lev, lna_gain_control, cco_current_lev,
  tx_freq_sep, rx_det_filter_tune, rx_if_filter_tune, tx_if_filter_tune, decision_level, rssi_ref, xo_cap,
  ref_r, reg_00000, r10001_compcode, r00001_appid1, r00010_appid2, r00011_appid3, r00100_appid4,
  r10101_channel, r10110_band, r10010_bias, r10011_atest, r10100_sens, wdog10111_xo, wdog11000_main,
  wdog11001_txxr, r11010wu_tune)
BEGIN
  CASE write IS
    WHEN '1' => IF (strobe = '1') THEN
      CASE addr_vect(addr_bus_width-3 DOWNTO 0) IS
        WHEN "00000" => data_out_temp <= reg_00000;
        WHEN "00010" => data_out_temp <= r00001_appid1;
        WHEN "00011" => data_out_temp <= r00100_appid2;
        WHEN "00101" => data_out_temp <= r00011_appid3;
        WHEN "00110" => data_out_temp <= r00110_appid4;
        WHEN "00111" => data_out_temp <= tx_power_lev;
        WHEN "01000" => data_out_temp <= lna_gain_control;
        WHEN "01001" => data_out_temp <= cco_current_lev;
        WHEN "01010" => data_out_temp <= tx_freq_sep;
        WHEN "01011" => data_out_temp <= rx_power_lev;
      END CASE;
    END IF;
    WHEN OTHERS => write <= '0';
  END CASE;
END PROCESS P1;
END para_data_inout_a;
WHEN "01011" => data_out_temp <= rx_if_filter_tune;
WHEN "01100" => data_out_temp <= tx_if_filter_tune;
WHEN "01101" => data_out_temp <= decision_level;
WHEN "01110" => data_out_temp <= rssi_ref;
WHEN "01111" => data_out_temp <= xo_cap;
WHEN "10000" => data_out_temp <= ref_r;
WHEN "10001" => data_out_temp <= r10001_compcode;
WHEN "10010" => data_out_temp <= r10010_bias;
WHEN "10011" => data_out_temp <= r10011_atest;
WHEN "10100" => data_out_temp <= r10100_sens;
WHEN "10101" => data_out_temp <= r10101_channel;
WHEN "10110" => data_out_temp <= r10110_band;
WHEN "10111" => data_out_temp <= wdog10111_xo;
WHEN "11000" => data_out_temp <= wdog11000_main;
WHEN "11001" => data_out_temp <= wdog11001_txrx;
WHEN "11010" => data_out_temp <= r11010wu_tune;
WHEN OTHERS => data_out_temp <= (OTHERS => 'Z');
END CASE;
ELSE
  data_out_temp <= (OTHERS => 'Z');
END IF;
WHEN OTHERS =>
  data_out_temp <= (OTHERS => 'Z');
END CASE;
END PROCESS P1;

P2: PROCESS (para_por_reset, strobe, read, addr_vect, data_in_out)
BEGIN
CASE para_por_reset IS
WHEN '0' =>
  r10001_compcode <= (OTHERS => '0');
  r00001_appid1 <= (OTHERS => '0');
  r00010_appid2 <= (OTHERS => '0');
  r00011_appid3 <= (OTHERS => '0');
  r0100_appid4 <= (OTHERS => '0');
  r10010_bias <= (OTHERS => '0');
  r10011_atest <= (OTHERS => '0');
  ant_tune <= "10000000";
  tx_power_lev <= "10000000";
  lna_gain_control <= "10000000";
  cco_current_lev <= "10000000";
  tx_freq_sep <= "10000000";
  rx_if_filter_tune <= "10000000";
  tx_if_filter_tune <= "10000000";
  decision_level <= "10000000";
  rssi_ref <= "10000000";
  xo_cap <= "10000000";
  ref_r <= "10000000";
  wdog10111_xo <= "10000000";
  wdog11000_main <= "10000000";
  wdog11001_txrx <= "10000000";
  r11010wu_tune <= "10000000";
WHEN OTHERS =>
  data_out_temp <= (OTHERS => 'Z');
  END CASE;
BEGIN
CASE addr_vect IS
WHEN "00000" => reg_00000 <= data_in_out;
WHEN "00001" => r00001_appid1 <= data_in_out;
WHEN "00010" => r00010_appid2 <= data_in_out;
WHEN "00011" => r00011_appid3 <= data_in_out;
WHEN "00100" => r00100_appid4 <= data_in_out;
WHEN "00101" => ant_tune <= data_in_out;
WHEN "00110" => tx_power_lev <= data_in_out;
WHEN "00111" => lna_gain_control <= data_in_out;
WHEN "01000" => cco_current_lev <= data_in_out;
WHEN "01001" => tx_power_lev <= data_in_out;
WHEN "01010" => tx_freq_sep <= data_in_out;
WHEN "01011" => tx_if_filter_tune <= data_in_out;
WHEN "01100" => rx_det_filter_tune <= data_in_out;
WHEN "01101" => rx_if_filter_tune <= data_in_out;
WHEN "01110" => rssi_ref <= data_in_out;
WHEN "01111" => xo_cap <= data_in_out;
WHEN "10000" => ref_r <= data_in_out;
WHEN "10001" => r10001_compcode <= data_in_out;
WHEN "10010" => r10010_bias <= data_in_out;
WHEN "10011" => r10011_atest <= data_in_out;
WHEN "10100" => r10100_sens <= data_in_out;
WHEN "10101" => wdog10111_xo <= data_in_out;
WHEN "10110" => wdog11000_main <= data_in_out;
WHEN "11000" => wdog11001_txrx <= data_in_out;
WHEN "11010" => r11010wu_tune <= data_in_out;
WHEN OTHERS =>
  data_out_temp <= (OTHERS => 'Z');
END CASE;
END IF;
WHEN OTHERS =>
  data_out_temp <= (OTHERS => 'Z');
END CASE;
END PROCESS P2;

- 70 -
END CASE;
END PROCESS P2;

P3: PROCESS (channel_in, para_ibs_in)
BEGIN
CASE para_ibs_in IS
  WHEN '1' =>
    r10101_channel <= channel_in((channelsel/2)-1 DOWNTO (channelsel/2)-data_bus_width);
    r10110_band((channelsel/2)-data_bus_width-1 DOWNTO 0) <= (OTHERS => '0');
  WHEN OTHERS =>
    END CASE;
END PROCESS P3;

P4: PROCESS (para_ibs_in)
BEGIN
CASE para_ibs_in IS
  WHEN '1' => nreset <= '0';
  WHEN OTHERS => nreset <= '1';
END CASE;
END PROCESS P4;
END rtl;

clk_div_a

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY clk_div_a IS
PORT( mwu_clk_in: IN STD_LOGIC;
      start_strobe: IN STD_LOGIC;
      in_por_reset: IN STD_LOGIC;
      decode_reset: IN STD_LOGIC;
      in_listen_strobe: IN STD_LOGIC;
      in_para_ibs_reset: IN STD_LOGIC;
      in_wu_en: IN STD_LOGIC;
      in_watch_dog: IN STD_LOGIC;
      out_ctrl_str: OUT STD_LOGIC;
      reset_out: OUT STD_LOGIC );
END clk_div_a;
ARCHITECTURE rtl OF clk_div_a IS
SIGNAL lna_strobe_sig2, str_alfa, mstrobe_out, reset_sig2, reset_sig3, reset_sig5, clk2Hz, clk4KHz: STD_LOGIC;
COMPONENT strobe_clk2_a
PORT( reset: IN STD_LOGIC;
      clk32kHz: IN STD_LOGIC;
      clk2Hz_out: OUT STD_LOGIC;
      clk4KHz_out: OUT STD_LOGIC )
END COMPONENT;
COMPONENT strobe_clk_a
PORT(clk4KHz_in: IN STD_LOGIC;
     clk2Hz_in: IN STD_LOGIC;
     strobe_out: OUT STD_LOGIC )
END COMPONENT;
BEGIN
lna_strobe_sig2 <= in_wu_en OR mstrobe_out;
str_alfa <= start_strobe OR lna_strobe_sig2;
out_ctrl_str <= str_alfa AND in_listen_strobe;
reset_sig5 <= decode_reset OR in_para_ibs_reset;
reset_sig3 <= reset_sig5 OR (NOT in_por_reset);
reset_sig2 <= NOT reset_sig3 OR in_watch_dog;
reset_out <= reset_sig2;
G1: strobe_clk2_a port map(clk2Hz_out => clk2Hz,
                           clk4KHz_out => clk4KHz,
                           reset => reset_sig2,
                           clk32kHz => mwu_clk_in);
G2: strobe_clk_a port map(clk4KHz_in => clk4KHz,
                           clk2Hz_in => clk2Hz,
                           strobe_out => mstrobe_out);
END rtl;
Appendix B

strobe_clk_a

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY strobe_clk_a IS
  PORT( clk4KHz_in: IN STD_LOGIC; --4KHz has time period 250us
         clk2Hz_in: IN STD_LOGIC; --2Hz has time period 500ms
         strobe_out: OUT STD_LOGIC --clk=1 every 500ms for 250us
  );
END strobe_clk_a;

ARCHITECTURE rtl OF strobe_clk_a IS

SUBTYPE count IS INTEGER RANGE 0 TO 2;
SIGNAL counter: count;

BEGIN
  P1: PROCESS(clk4KHz_in, clk2Hz_in, counter)
  BEGIN
    IF (clk2Hz_in='0') THEN
      counter<=0;
      strobe_out<='0';
    ELSIF (RISING_EDGE(clk4KHz_in)) THEN
      CASE counter IS
      WHEN 0 => strobe_out<='1';
      counter<=counter+1;
      WHEN OTHERS => counter<=counter;
      END CASE;
    END IF;
  END PROCESS P1;
END rtl;

strobe_clk2_a

This block was designed manually in Cadence and therefore there is no source code written in VHDL or Verilog. This clock divider was designed because it has lower power consumption compared to a standard divider made in VHDL. The output of this block is the input to the strobe_clk_a which create the strobe signal ctrl_str.

---

Figure 1 Extract from Cadence showing the block strobe_clk_a
The testbenches that were designed and used during the verification phase in simulations were written in Verilog. The same testbench was used, with minor changes, for showing that the logic worked as wanted when different scenarios appeared. In the testbench wake_ctrl_testb, there are comments showing where these changes were made.

**wake_ctrl_testb:**
```verilog
timescale 1ns/1ns
module wake_ctrl_testb;
reg wake_sig, wake_data, mac_ready, wu_clk, wu_en, ibs_in, str, rw, por_reset, watch_dog;
reg [6:0] addr;
reg [7:0] data_in;
wire vreg_en, osc_en, ctrl_str;
wire [7:0] data_inout, bias_vect, testio_ctrl, sens_vect;
assign data_inout = data_in;

wake_ctrl_a DUT ( wake_sig, wake_data, mac_ready, wu_clk, wu_en, ibs_in, str, rw, por_reset, watch_dog,
                addr, data_inout, vreg_en, osc_en, ctrl_str, bias_vect, sens_vect, testio_ctrl );

initial
begin
    por_reset=1'b0; //por_reset='0'
    wu_en=1'b0;  //wakeup_enable, direct control over lna_str and det_str
    addr=7'b0000000; //address from MAC used the bus system
    mac_ready=1'b0; //mac_ready signal from MAC
    #525
    por_reset=1'b1; //por_reset='1'
    wake_data=1'b1; //simulated RF data clocked in with wake_sig simulated further down
    #50
    wake_data=1'b0;
    #50
    wake_data=1'b1;
    #50
    wake_data=1'b1;
    #50
    wake_data=1'b1;
```

Appendix C

```verilog
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
//ibs_in=1'b0; //used in testbench when testing the ibs pin function. See figure 11
#50
wake_data=1'b0;
#50
// watch_dog=1'b1; //watchdog ='1', go back to sleep mode, used when testing ibs pin
#50
wake_data=1'b1;
#50
// watch_dog=1'b0; //watchdog ='1', go back to sleep mode, used when testing ibs pin
#50
// ibs_in=1'b1; //used in testbench when testing the ibs pin function
#50
wake_data=1'b0;
#50
//start sequence begin
#50
wake_data=1'b1;
#50
wake_data=1'b1;
#50
wake_data=1'b1;
#50
wake_data=1'b1;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b0; //start sequence at end
#50
wake_data=1'b1;
#50
//0 First bit in companycode & codecheck
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
```

- 74 -
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;    //7
#50
wake_data=1'b1;    //8
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;    //15 Last bit in companycode
#50
wake_data=1'b1;    //16 first bit in Device ID or Master ID
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b0;    //16
#50
wake_data=1'b0;    //17
#50 wake_data=1'b1;
#50 wake_data=1'b1;
#50 wake_data=1'b0;
#50 wake_data=1'b0;
#50 wake_data=1'b1;
#50 wake_data=1'b1;
#50 wake_data=1'b0;
#50 wake_data=1'b0;
#50 wake_data=1'b1;
#50 wake_data=1'b1;
#50 wake_data=1'b0;
#50 wake_data=1'b0;
#50 wake_data=1'b1;
#50 wake_data=1'b1;
#50 wake_data=1'b0;
#50 wake_data=1'b0;
#50 wake_data=1'b1;
#50 wake_data=1'b0;   //68 Last bit in Device ID/Master ID
#50 wake_data=1'b0;   //69 first bit in channel ID
#50 wake_data=1'b1;
#50 wake_data=1'b1;
#50 wake_data=1'b0;
#50 wake_data=1'b0;
#50 wake_data=1'b1;
#50 wake_data=1'b1;
#50 wake_data=1'b0;
#50 wake_data=1'b0;
#50 wake_data=1'b1;
#50 wake_data=1'b1;
#50 wake_data=1'b0;
#50 wake_data=1'b0;
#50 wake_data=1'b1;
#50 wake_data=1'b1;
#50 wake_data=1'b0;
#50 wake_data=1'b0;
#50 wake_data=1'b1;
#50 wake_data=1'b1;
#50 wake_data=1'b0;
#50 wake_data=1'b0;
#50 wake_data=1'b1;
#50 wake_data=1'b1;
#50 wake_data=1'b0;
#50 wake_data=1'b0;
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b1;  // 92 last bit in channelsel
#50
wake_data=1'b1;  // wakein_decode shall not shift in any more data
#50
mac_ready=1'b1;  // mac_ready sets high (assumed that correct packet was detected)
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
data_in=8'b10000001;  // data is set to the data_inout bus
rw=1'b0;  // read/write is set to 0 = write
addr=7'b1100000;  // address "00000" => reg_00000;
#50
str=1'b1;  // strobe sets high and para_data_inout handles the communication on the bus
#50
str=1'b0;  // strobe sets low and para_data_inout stops the communication on the bus
data_in=8'bz;
#100
data_in=8'b10100001;
rw=1'b0;
addr=7'b1100000;  // address "00001" => r00001_appid1;
#50
str=1'b1;
#50
str=1'b0;
data_in=8'bz;
#100
data_in=8'b10101001;
rw=1'b0;
Appendix C

```verbatim
addr=7'b1100010;  // "00010" => r00010_appid2;
  #50
str=1'b1;
  #50
str=1'b0;
  #50
data_in=8'b0z;
  #100
data_len=8'b10110001;
rw=1'b0;
addr=7'b1100011;  // "00011" => r00011_appid3;
  #50
str=1'b1;
  #50
str=1'b0;
  #50
data_in=8'b0z;
  #100
data_len=8'b101110001;
rw=1'b0;
addr=7'b1100100;  // "00100" => r00100_appid4;
  #50
str=1'b1;
  #50
str=1'b0;
  #50
data_in=8'b0z;
  #100
data_len=8'b11001100;
rw=1'b0;
addr=7'b1110010;  // "10010" => r10010_bias
  #50
str=1'b1;
  #50
str=1'b0;
  #50
data_in=8'b0z;
  #100
data_len=8'b11000110;
rw=1'b0;
addr=7'b1110011;  // "10011" => r10011_atest
  #50
str=1'b1;
  #50
str=1'b0;
  #50
data_in=8'b0z;
  #100
data_len=8'b11011110;
rw=1'b0;
addr=7'b1110100;  // "10100" => r10100_sens
  #50
str=1'b1;
  #50
str=1'b0;
  #50
data_in=8'b0z;
  #100
data_len=8'b110100001;
rw=1'b0;
addr=7'b1110101;  // "10101" => ant_tune
  #50
str=1'b1;
  #50
str=1'b0;
  #50
data_in=8'b0z;
  #100
data_len=8'b110101010;
rw=1'b0;
addr=7'b1110110;  // "10110" => tx_power_lev
  #50
str=1'b1;
  #50
str=1'b0;
  #50
data_in=8'b0z;
  #100
data_len=8'b111000000;
rw=1'b0;
addr=7'b1110111;  // "10111" => lna_gain_control
  #50
str=1'b1;
  #50
str=1'b0;
  #50
data_in=8'b0z;
  #100
data_len=8'b111100000;
rw=1'b0;
addr=7'b11101000;  // "01000" => cco_current_lev
  #50
str=1'b1;
  #50
str=1'b0;
  #50
data_in=8'b0z;
  #100
data_len=8'b111010000;
rw=1'b0;
```

- 78 -
str=1'b0;
data_in=8'bz;
#100
data_in=8'b11111100;
rw=1'b0;
addr=7'b1101001;       // "01001" => tx_freq_sep
#50
str=1'b1;
#50
str=1'b0;
data_in=8'bz;
#100
data_in=8'b11111110;
rw=1'b0;
addr=7'b1101010;       // "01010" => rx_det_filter_tune
#50
str=1'b1;
#50
str=1'b0;
data_in=8'bz;
#100
data_in=8'b11111110;
rw=1'b0;
addr=7'b1101101;       // "01101" => rx_if_filter_tune
#50
str=1'b1;
#50
str=1'b0;
data_in=8'bz;
#100
data_in=8'b11111110;
rw=1'b0;
addr=7'b1101111;       // "01111" => xo_cap
#50
str=1'b1;
#50
str=1'b0;
data_in=8'bz;
#100
data_in=8'b11111110;
rw=1'b0;
addr=7'b1110001;       // "10000" => ref_r
#50
str=1'b1;
#50
str=1'b0;
data_in=8'bz;
#100
data_in=7'b1110000;
rw=1'b0;
addr=7'b1110001;       // "10001" => r10001_compcode
#50
str=1'b1;
#50
str=1'b0;
data_in=8'bz;
#100
data_in=8'b11111001;
rw=1'b0;
addr=7'b1110001;       // "10000" => ref_r
#50
str=1'b1;
#50
str=1'b0;
data_in=8'bz;
#100
data_in=8'b11111101;
rw=1'b0;
addr=7'b1110001;       // "10000" => ref_r
#50
str=1'b1;
#50
str=1'b0;
data_in=8'bz;
#100
//the register that is addressed puts out on the data_inout bus

addr=7'b100000; #50
str=1'b1; #50
str=1'b0; #50
addr=7'b100001; #50
str=1'b1; #50
str=1'b0; #50
addr=7'b1100101; #50
str=1'b1; #50
str=1'b0; #50
addr=7'b1100110; #50
str=1'b1; #50
str=1'b0; #50
addr=7'b1100111; #50
str=1'b1; #50
str=1'b0; #50
addr=7'b1101000; #50
str=1'b1; #50
str=1'b0; #50
addr=7'b1101001; #50
str=1'b1; #50
str=1'b0; #50
addr=7'b1101010; #50
str=1'b1; #50
str=1'b0; #50
addr=7'b1101011; #50
str=1'b1; #50
str=1'b0; #50
addr=7'b1101100; #50
str=1'b1; #50
str=1'b0; #50
addr=7'b1101101; #50
str=1'b1; #50
str=1'b0; #50
addr=7'b1101110; #50
str=1'b1; #50
str=1'b0; #50
addr=7'b1101111; #50
str=1'b1; #50
str=1'b0; #50
addr=7'b1110000; #50
str=1'b1; #50
str=1'b0; #50
str=1'b0; #50
addr=7'b1110001;
addr=7'b1110010;
addr=7'b1110011;
addr=7'b1110100;
addr=7'b1110110;
addr=7'b0000000;
mac_ready=1'b0; //mac_ready set low, go back to sleep mode
wake_data=1'b1; //new rf signal
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;
wake_data=1'b0;
wake_data=1'b1;

Appendix C

```vhdl
#50 wake_data=1'b0;
#50 wake_data=1'b1;
#50 wake_data=1'b0;
#50 wake_data=1'b1;
#50 wake_data=1'b0;
#50 wake_data=1'b1;
#50 wake_data=1'b0;
#50 //start sequence begin
#50 wake_data=1'b1;
#50 wake_data=1'b1;
#50 wake_data=1'b1;
#50 wake_data=1'b1;
#50 wake_data=1'b0;
#50 wake_data=1'b0;
#50 wake_data=1'b0;
#50 //start sequence at end
#50 wake_data=1'b0;
#50 //0 First bit in companycode, this time the spec_companycode
#50 wake_data=1'b0;
#50 // that was read in during the first RF signal
#50 wake_data=1'b1;
#50 wake_data=1'b0;
#50 wake_data=1'b0;
#50 wake_data=1'b1;
#50 wake_data=1'b1;
#50 wake_data=1'b0;
#50 wake_data=1'b1;
#50 //7
#50 wake_data=1'b0;
#50 //8
#50 wake_data=1'b1;
#50 wake_data=1'b0;
#50 wake_data=1'b0;
#50 wake_data=1'b1;
#50 wake_data=1'b1;
#50 wake_data=1'b0;
#50 wake_data=1'b1;
#50 //15 Last bit in companycode
#50 wake_data=1'b0;
#50 //16 first bit in Device ID that was read in during the
#50 // first RF signal
#50 wake_data=1'b1;
#50 wake_data=1'b1;
#50 wake_data=1'b1;
#50 wake_data=1'b0;
#50 wake_data=1'b1;
#50 wake_data=1'b1;
#50 wake_data=1'b0;
#50 //this bit is changed when testing the Manchester code! See figure 8
#50 wake_data=1'b1;
#50 //together with the changed bit above this bit is changed
#50 //and it is then wrong application ID, but ok Manchester code! See figure 9
```
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
//ibs_in=1'b0; //used in testbench when testing the ibs pin function. See figure 11.
#50
wake_data=1'b1;
#50
wake_data=1'b1;
#50
mac_ready=1'b1; //mac_ready sets high (assumed that MAC nad RF part started correct)
//used when testing ibs pin
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
mac_ready=1'b0; //mac_ready sets low (assumed that MAC wants to power down)
//used when testing ibs pin
wake_data=1'b1;
#50
//ibs_in=1'b1; //used in testbench when testing the ibs pin function
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b1;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b1;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b1;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b1;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b1;
#50
wake_data=1'b0; //68 Last bit in Device ID
#50
wake_data=1'b1; //69 first bit in channel ID
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1; //92 last bit in channel ID
#50
wake_data=1'b1; //following data shall not be shifted in
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
#50
wake_data=1'b1;
#50
wake_data=1'b0;
// #500020 //this prolongs the simulation... needed for test of strobe-extraction
// #3980 //these bits are needed for testing the wu_en signal. See figure 10 further down.
// #5980 //wu_en=1'b1;
// #5020 //wu_en=1'b0;
// #485000 #50
//wu_en=1'b1;
#$finish;
end
always
begin
#5
wu_clk=1'b1; //incoming clock signal from application (32kHz in this implementation)
#5
wu_clk=1'b0;
end

always
begin
#25
wake_sig=1'b1; //incoming RF signal used as clocksignal
#25
wake_sig=1'b0;
end

initial
begin
dumpfile("wake_ctrl_a_dump.vcd");
dumpvars;
end
endmodule