BACHELOR’S THESIS

FPGA IMPLEMENTATION OF NFC TYPE 2
PICC

Thesis author: Tore Leikanger

Thesis supervisor: Juha Häkkinen

August, 2016
In this text the implementation of an NFC Forum compatible type 2 PICC on an FPGA is described. The description begins with an introduction to the technology, followed by a brief overview of the development module. The last part is about the actual implementation, and testing of this. The implementation is done in SystemVerilog, and verified on an FPGA platform with NFC analog front end and antenna on the circuit board. The motivation of this project is to make a working NFC-A PICC, for further investigation and exploration about the possibilities behind this technology, and how these can be exploited to for example develop sensor and measurement systems.

Keywords: NFC, PICC, FPGA, SystemVerilog
TIIVISTELMÄ


Avainsanat: NFC, PICC, FPGA, SystemVerilog
# TABLE OF CONTENTS

ABSTRACT
TIIVISTELMÄ
TABLE OF CONTENTS
LIST OF ABBREVIATIONS AND SYMBOLS

1 INTRODUCTION ................................................................. 6

2 BASIC PRINCIPLES OF NFC FUNCTIONALITY ................. 7
   2.1 Inductive coupling ..................................................... 7
   2.2 Bit-level encoding .................................................... 7
   2.3 NFC frames ............................................................. 8
   2.4 NFC frame delay time .............................................. 9
   2.5 Cyclic redundancy check in NFC-A ......................... 10
   2.6 NFC Type 2 commands and states ......................... 11
   2.7 NFC Type 2 memory map .................................. 13

3 THE NFC FORUM TYPE 2 PICC DESIGN ...................... 15
   3.1 The analog front end ............................................. 15
   3.2 The digital RX demodulator ................................. 16
   3.3 The digital TX modulator .................................. 17
   3.4 The NFC Type 2 core ......................................... 21

4 TESTING THE PICC FUNCTIONALITY .......................... 23

5 DISCUSSION .............................................................. 25

6 SUMMARY ................................................................. 27

REFERENCES ............................................................... 28

ATTACHMENTS ............................................................. 29
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACK</td>
<td>Acknowledge</td>
</tr>
<tr>
<td>ATQA</td>
<td>Answer-to-Request (NFC-A)</td>
</tr>
<tr>
<td>CRC</td>
<td>Cyclic Redundancy Check</td>
</tr>
<tr>
<td>EEPROM</td>
<td>Electrically Eraseble Programmable Read-Only Memory</td>
</tr>
<tr>
<td>EoF</td>
<td>End of Frame</td>
</tr>
<tr>
<td>etu</td>
<td>Elementary time unit</td>
</tr>
<tr>
<td>FDT</td>
<td>Frame Delay Time</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field-Programmable Gate Array</td>
</tr>
<tr>
<td>I2C</td>
<td>Inter-Integrated Circuit</td>
</tr>
<tr>
<td>lsb</td>
<td>Least significant bit</td>
</tr>
<tr>
<td>LSB</td>
<td>Least Significant Byte</td>
</tr>
<tr>
<td>MSB</td>
<td>Most Significant Byte</td>
</tr>
<tr>
<td>NAK</td>
<td>Negative acknowledge</td>
</tr>
<tr>
<td>NFC</td>
<td>Near Field Communication</td>
</tr>
<tr>
<td>NRZ</td>
<td>Non-Return to Zero</td>
</tr>
<tr>
<td>PCB</td>
<td>Printed Circuit Board</td>
</tr>
<tr>
<td>PCD</td>
<td>Proximity Coupling Device</td>
</tr>
<tr>
<td>PICC</td>
<td>Proximity Inductive Coupling Cards</td>
</tr>
<tr>
<td>REQA</td>
<td>Request (NFC-A)</td>
</tr>
<tr>
<td>SoF</td>
<td>Start of Frame</td>
</tr>
<tr>
<td>UID</td>
<td>Unique Identifier</td>
</tr>
<tr>
<td>WUPA</td>
<td>Wake-Up (NFC-A)</td>
</tr>
</tbody>
</table>

n The constant defining the length of FDT
1. INTRODUCTION

Near field communication (NFC) standards specify a wireless communication technology for very short range applications, typically less than 10 cm [1]. The standards in this technology are specified in the ISO/IEC 14443 documents and NFC Forum documents. The NFC Forum standards has originated from the radio frequency identification (RFID) standards, described in the ISO/IEC 14443 documents (1-4).

NFC type 2 proximity inductive coupling cards (PICC, the NFC tag) are a subset of the standard, using NFC-A technology [2]. The PICC type specifies the technology used, as well as the memory mapping in the device and the minimum set of available commands [2]. More commands are however possible to implement, and are often implemented to give PICCs more sophisticated functions. NFC type 2 is one of the most commonly used type, both for ticketing, key cards, and NFC posters.

Recently, the use of NFC PICCs as an interface between an NFC reader (e.g. a smart phone) and a microcontroller or a sensor has been explored by many, and commercial products featuring this possibility has emerged [3]. Usually the NFC chip features an inter-integrated circuit (I2C) slave controller. The EEPROM in the NFC chip can thus be read from or written to both through NFC and I2C, these devices are called dual interface EEPROMs. Another approach is to implement a microcontroller in the NFC chip.

In this text, an NFC type 2 core implemented with SystemVerilog is presented. Also, a printed circuit board (PCB) with the NFC analog front end designed with discrete components connected to a field-programmable gate array (FPGA) is presented. Critical parts of the SystemVerilog code will also be shown, as well as measured wave forms at critical nodes on the analog front end.
2. BASIC PRINCIPLES OF NFC FUNCTIONALITY

2.1. Inductive coupling

NFC is working through inductive coupling between matched resonant LC-circuits. When the matched inductors are close enough, alternating current in the proximity coupling device (PCD, the NFC reader) inductor induces current in the tag coil through an electromagnetic field between the inductor antennas. In NFC technologies, this electromagnetic field has a carrier frequency of 13.56 MHz, and this is amplitude modulated for PCD-to-PICC data transfer. This amplitude modulation is possible to detect on the tag side by using an envelope detector. For PICC-to-PCD data transfer, the tag is modulating the load on the LC antenna circuit, by switching on and off either a capacitive or a resistive load. The extra capacitive load works such that the tag antenna no longer is matched with the field, while the resistive load works such that the tag antenna is reflecting a bigger part of the energy received. In both cases, part of the field emitted by the PCD is scattered back from the tag antenna, and thus detectable. [1]

2.2. Bit-level encoding

In the NFC Forum standards, six patterns are described for bit encoding patterns, three for PCD to PICC data transfer, and three for PICC to PCD data transfer [4]. These bit patterns can be seen in fig. 1, and are described more closely below.

![Figure 1: The bit patterns in NFC-A technologies.](image)

The data transfer rate in NFC technologies as described in the NFC Forum
standards [4] is typically 106 kbps (13.56 MHz / 128). In some NFC types described in the standards, the data transfer rate can be increased to 212 kbps, 424 kbps or 848 kbps. The duration of one bit is called one elementary time unit (etu), which is about 9.44 µs for a 106 kbps transfer rate.

In PCD to PICC transfer, the bit patterns X, Y and Z are defined. These are defined as follows: The pattern is X if a pulse is detected beginning at the center of an etu, Z if a pulse is detected at the beginning of an etu, and Y if no pulse is detected during one etu. If the X pattern is detected, this is always interpreted as logical 1, while both patterns Y and Z are interpreted as logical 0. After an X pattern, logical zero has pattern Y, otherwise pattern Z. This encoding scheme is called the Modified Miller encoding, and is used in all NFA-A types. All communication frames are synchronized by pattern Z, which is the start of frame (SoF). The end of frame (EoF) is recognized by two logical zeros where the last is with pattern Y, i.e. patterns ZY or YY depending on the preceding bit. [4]

In PICC to PCD transfer, the bit patterns D, E and F are defined. Here, pattern D is defined as logical 1 and is recognized by four 848 kHz pulses in the first half of the etu. Pattern E is defined as logical 0 and is recognized by four 848 kHz pulses in the second half of the etu. Pattern F is only used for EoF. As can be seen, pattern D and E are Manchester encoded on top of an 848 kHz sub-carrier. For faster bit rates, the number of 848 kHz pulses during one bit is reduced. The PICC to PCD synchronizing SoF is defined to be pattern D (logical 1), and EoF is defined as pattern F. [4]

2.3. NFC frames

In NFC technologies, all transmissions are least-significant bit (lsb) first, and standard frames are divided into 8-bit bytes, shown in figure 2. Every frame is synchronized/initialized with a SoF, and ended by an EoF, and every byte in the frame is followed by an odd parity bit. The exceptions to these rules are the handshake requests and anti-collision frames, which are done using short frames and anti-collision frames respectively. Also acknowledge/no acknowledge (ACK/NAK) bytes from PICC to PCD are short frames. Short frames, shown
in fig. 3, are shorter than standard frames, i.e. with a maximum of 7 bits, and have no parity bit. Anti-collision frames, shown in fig. 4, are similar to standard frames in bit number and are using parity bits, but these can be ended at any bit during the frame by the PCD, from where the PICC must continue the frame. Also, standard frames have two cyclic redundancy check (CRC) bytes added to the end of the frame for transmission error check. [2, 4]

![Figure 2: Standard frame as defined in the NFC-A standard.](image)

Figure 2: Standard frame as defined in the NFC-A standard.

![Figure 3: A 7-bit (left) and 4-bit (right) short frame, as defined in the NFC-A standard.](image)

Figure 3: A 7-bit (left) and 4-bit (right) short frame, as defined in the NFC-A standard.

![Figure 4: An anti-collision frames as defined in the NFC-A standard. Notice how the frame is shared between the PCD and PICC.](image)

Figure 4: An anti-collision frames as defined in the NFC-A standard. Notice how the frame is shared between the PCD and PICC.

### 2.4. NFC frame delay time

The frame delay time (FDT) between the PCD and PICC frames are accurately defined, and are dependent on the last bit of the PCD frame. The delay time between the PICC and the PCD frames are not however as accurately defined [4]. A graphical description of the PCD to PICC FDT is shown in fig. 5.

The PCD to PICC FDT is defined as

\[
FDT_0 = n \text{etu} + \frac{20}{f_c},
\]

if the last bit of the PCD frame is logical 0 and

\[
FDT_1 = n \text{etu} + \frac{84}{f_c},
\]

if the last bit of the PCD frame is logical 1.
if the last bit of the PCD frame is logical 1. Here, \( n \) is the number of etus between the last rising edge of the PCD signal and the first edge of the PICC SoF. It is noteworthy that the difference between \( FDT_0 \) and \( FDT_1 \) is \( 64 \cdot f_c \), which one half of an etu. The values of \( n \) in NFC-A type 2 NFC PICCs are 9 for handshake and anti-collision requests (WUPA, ATQA and anti-collision requests), and only defined as larger than 9 for all other requests. The maximum value of \( n \) is defined by the PCD. [4]

### 2.5. Cyclic redundancy check in NFC-A

The cyclic redundancy check (CRC) defined in the NFC-A standard [2, 4] is a 16-bit CRC with polynomial \( x^{15} + x^{12} + x^5 + 1 \) and initial value of \( 6363h \). The data is shifted into the CRC bit by bit, lsb first, and neither the input nor the output are inverted. The residue of the CRC generator, i.e. the value the CRC should have when both the data and the CRC bytes are shifted into the CRC generator, is \( 0000h \). The schematic of the CRC generator can be seen in fig. 6. The 16-bit outputs of the CRC generator are at the outputs of the d-flip-flops.
2.6. NFC Type 2 commands and states

For all NFC technologies, the NFC PCD is the master of the communication, and executes all the requests. The PICC is the slave, i.e. the PICC can only answer to the requests, and may never initiate a transmission of data. This enables the PCD to completely control the communication in the field, by communicate with several PICCs at the same time. More than one PCD in the field at the same time is however not possible in NFC-A technologies. [2, 4]

When an NFC-A PICC enters the readers magnetic field, the PICC initiated in the IDLE state. As can be seen in fig. 7, the PICC can only leave the IDLE state when the NFC-A request (REQA) or wake-up (WUPA) requests are received. If the PICC is in the HALT state, it only reacts to the WUPA request, but otherwise behaves the same way as in the IDLE state. The WUPA and ATQA frames are 7-bit short frames, with no parity or CRC. The PICC answers to these requests with the NFC-A answer-to-request (REQA) answer. REQA includes information about the tag type (type 1, 2 or 4) and the size of the unique identifier (UID). The UID size can be 4 bytes, 7 bytes or 10 bytes. When the PICC has answered with ATQA, it enters the READY 1 state. Now the PICC in fig. 7 is an NFC-A PICC with 7 UID bytes, because only two READY states are present. Only one READY state is present if the PICC has 4 UID bytes, and 3 READY states for 10 UID bytes. The reason for this is the length of the anti-collision and select requests, where a maximum of 4 UID bytes can be given for each request. [2, 4]

In the READY 1 and READY 2 states, the PICC is confronted with the anti-collision request. The anti-collision request use anti-collision frames, which are shared frames with the PCD and the PICC. This frame includes the request identifier as well as information about where the breakpoint, i.e. where the PCD stops sending the frame and the PICC should continue. This breakpoint can be at any point after the second byte and before the last bit of the frame. In addition to the request identifier and the position of the break point, the anti-collision frame includes the UID bytes, a byte count check (BCC) byte, and an indication whether more cascade levels (i.e. more READY states) are needed to transmit all UID bytes. In addition, the PCD can skip the anti-collision sequence, by
Figure 7: The states and the transitions between the states in an NFC type 2 PICC.

requesting a read from address 0. When the PCD has received the whole anticollision frame, the PCD sends a select request, to which the PICC should answer with its SAK answer. The value of the SAK address also indicates whether the PICC enters the ACTIVE state or the next READY state. [2, 4]

What is not shown in fig. 7, is that if another command than the anti-collision request, select request or the read from address 0 request is detected while the PICC is in a READY state, the PICC should return to its initial state (IDLE or HALT). [4]

When the PICC is in the ACTIVE state, the main requests are the READ and WRITE requests. In addition to these, the PCD can set the PICC into the HALT state with the HALT request. The READ and WRITE requests are reading or writing to the actual EEPROM in the NFC type 2 tag. The READ request is reading 16 bytes (i.e. four 4-byte blocks), and the WRITE request in writing 4 bytes (i.e. one 4-bytes block). [5]
2.7. NFC Type 2 memory map

The memory map of NFC type 2 PICCs are either static or dynamic. The static
memory has up to 64 bytes memory, of which 48 bytes are for user data. The
dynamic memory mappings are for memories with more than 64 bytes. [5]

All NFC type 2 PICCs have the memory divided into blocks with 4 bytes in
each block. These are such that if a PCD is writing to a NFC type 2 memory,
one whole block has to be written to. Also, if the PCD is reading from an NFC
type 2 memory, 4 whole blocks are read. Accessing single bytes within a block is
not possible. Further, the address given in a read or write request by the PCD is
the address to the block. In fig. 8 the most basic NFC type 2 memory layout is
shown, with 48 data bytes in blocks 0x04 to 0x0f, and 4 blocks for UID bytes and
lock bytes in blocks 0x00 to 0x03. The dynamic memory map is similar to this,
but with more user data blocks. Also, a dynamic memory map includes more
lock bytes, which may be at any higher byte. The highest possible block address
in a type 2 NFC PICC is 0xFF, i.e. the memory includes a maximum of 1 kB
of memory. To further increase the memory, it is possible to divide the memory
into more sectors. The PCD then choose a sector before a read or write request
is sent. [5]

<table>
<thead>
<tr>
<th>Block</th>
<th>Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>UID BYTES</td>
</tr>
<tr>
<td>0x01</td>
<td>UID BYTES</td>
</tr>
<tr>
<td>0x02</td>
<td>UID BYTES</td>
</tr>
<tr>
<td>0x03</td>
<td>LOCK BYTES</td>
</tr>
<tr>
<td>0x04</td>
<td>CAPABILITY CONTAINER</td>
</tr>
<tr>
<td>0x05</td>
<td>USER DATA</td>
</tr>
<tr>
<td>0x06</td>
<td></td>
</tr>
<tr>
<td>0x07</td>
<td></td>
</tr>
<tr>
<td>0x08</td>
<td></td>
</tr>
<tr>
<td>0x09</td>
<td></td>
</tr>
<tr>
<td>0x0A</td>
<td></td>
</tr>
<tr>
<td>0x0B</td>
<td></td>
</tr>
<tr>
<td>0x0C</td>
<td></td>
</tr>
<tr>
<td>0x0D</td>
<td></td>
</tr>
<tr>
<td>0x0E</td>
<td></td>
</tr>
<tr>
<td>0x0F</td>
<td></td>
</tr>
</tbody>
</table>

Figure 8: The memory mapping used in the design, which is the most basic
memory mapping for NFC type 2 PICCs.
The FPGA based PICC presented in this text however use the most basic static memory map shown in fig. 8, and the dynamic memory model will not be further explored in this text.

The UID bytes in blocks 0x00 to 0x02 includes not only the UID bytes, but also the byte count check (BCC) bytes used in the anti-collision sequence. The lock bytes are telling which of the blocks are read only. Further, the capability container (CC) gives information about the size of the memory, which version of the NFC Forum standard is used, and the read/write capability of the tag.
3. THE NFC FORUM TYPE 2 PICC DESIGN

The circuit is built around the Altera MAX 10 FPGA, with a discrete component analog front end (AFE). 16 general purpose FPGA IO pins are broken out for debugging and further development. A picture of the device is shown in fig. 9.

![Figure 9: The NFC development device used to develop the NFC type 2 PICC.](image)

3.1. The analog front end

The schematics of the analog layout can be seen in fig. 10. The load modulation of the output is also seen at the input because the envelope detector sees this as amplitude modulation. In the NFC field the load modulation is seen as pulses of stronger field, thus the name back-scattering. The demodulator circuit is strongly inspired by the Chameleon mini project [6].

![Figure 10: The circuit diagram of the analog front end.](image)
3.2. The digital RX demodulator

When the NFC signal enters the FPGA from the envelope detector in the analog front end, the signal is Modified Miller modulated, as is described in section 2.2. Because the lengths of the pulses are not defined more accurately than shorter than the half of an etu, the rising and falling edges of the pulse can be listened to. If a rising edge is received during the first half of the etu, the X pattern is received. Similarly, the Y pattern is received if no rising edges are received, and the Z pattern if the rising edge is received during the second half of the etu. For this demodulation scheme to work, the serial clock must be synchronized at the SoF bit, which is the X pattern. The SystemVerilog code to achieve this functionality is shown below.

```verilog
// Data-in Edge Listening
always_ff @ (posedge clk or negedge rst_n)
begin
  if ( rst_n == '0 )
    din_sync <= '1;
  else if ( state == POFF )
    din_sync <= '1;
  else
    din_sync <= {din_sync[1:0], (DIN | disable_rx)};
end

always_comb begin
  din_fe = din_sync[2] & ~din_sync[1];
  din_re = ~din_sync[2] & din_sync[1];
end

// Pattern Recognition
logic [1:0] prev_pattern;
// Check for pulse
always_ff @ (posedge clk or negedge rst_n)
begin
  if ( rst_n == '0 )
    received_re <= '0;
  else if ( state == POFF )
    received_re <= '0;
  else if ( clk106 == 1 )
    received_re <= '0;
  else if ( clk106_n == 1 )
    received_re <= '0;
  else if ( din_re == 1 )
    received_re <= '1;
```
When the demodulation is done as shown above, the bit value is shifted into a 8
bit shift register. The byte is stored in a buffer when completely received, such
that the circuit can respond to the request when the frame is done. SoF is found
by listening for the Z pattern, and EoF is found by listening for Y after a logical
0.

3.3. The digital TX modulator

The NFC PICC to PCD communication is Manchester encoded on an 848 kHz
sub-carrier as described in section 2.2. This is done by shifting out a data,
and then modulating this by using an “exclusive or” and an “and” port. The
modulation of the signal, and the transmitter shift register control is done by the
following SystemVerilog codes

```
// Modulating the data out signal
always_comb DOUT = ~nfca_tx_done & carrier_on & clk848
& (nfca_tx ^ clk106_50p);
```

Now, nfca_tx_done, carrier_on, clk848 and clk106_50p are signals to control the
modulation of the signal, while the nfca_tx is a non-return-to-zero (NRZ) signal.
nfca_tx_done is defined to prevent an additional spike after the frame is sent, carrier_on defining when the load-modulation is active, clk848 is the 50 % duty-cycle sub-carrier used in the modulated signal, and clk106.50p is the clock signal used for manchester encoding. The carrier_on signal is also used for disable the RX logics above to avoid the transmission to be disturbed by the synchronization features in the RX logics.

The nfca_rx signal is generated by the module below in NRZ format, i.e. each byte is packed and shifted out lsb-first. Further, SoF and EoF are generated, and the parity bit is automatically generated and placed between every byte.

```verilog
module nfca_tx_ctrl(  // System signals
  input clk ,
  input rst_n ,
  // Buffer control
  input [7:0] data ,
  output logic data_latched ,
  // tx line control and line
  input init_tx ,
  input more_tx ,
  input [2:0] first_byte_n ,
  input sclk ,
  output logic tx ,
  output logic tx_done ) ;

logic [7:0] tx_buff , nxt_tx_buff ;
logic [3:0] tx_counter ;
logic buff_update , buff_latch , par_update ;
logic sofdone , pardone , par , eofflag ;
enum logic [2:0] {SOF , PAR , TX_ACTIVE , INACTIVE} tx_state , nxt_tx_state ;
always_comb tx_done = eofflag & sclk & tx_state == INACTIVE ;
always_comb nxt_tx_buff = {1'b0 , tx_buff [7:1]} ;
always_ff @ ( posedge clk or negedge rst_n )
begin
  if ( rst_n == '0 )
  begin
    tx_buff <= '0 ;
    data_latched <= '0 ;
    eofflag <= '0 ;
  end
  else if ( data_latched == '1 )
    data_latched <= '0 ;
  else if ( buff_update == '1 )
  begin
    tx_buff <= nxt_tx_buff ;
  end
end
```
else if (buff_latch == '1')
begin
  if (more_tx)
  begin
    tx_buff <= data;
    data_latched <= '1;
  end
else
  eofflag <= '1;
end
else if (tx_done)
eofflag <= '0;
end

always_comb
begin
  if (tx_state == INACTIVE & init_tx == '1)
  begin
    nxt_tx_state = TX_ACTIVE; // PAR;
  end
else if (tx_state == TX_ACTIVE & (tx_counter[3] == '1))
  begin
    nxt_tx_state = PAR;
  end
else if (tx_state == PAR & pardone == '1)
  begin
    if (eofflag)
      nxt_tx_state = INACTIVE; // EOF = no modulation
    else
      nxt_tx_state = TX_ACTIVE;
  end
else
  nxt_tx_state = tx_state;
end

always_ff @ (posedge clk or negedge rst_n)
begin
  if (rst_n == '0)
    tx_state <= INACTIVE;
  else
    tx_state <= nxt_tx_state;
end

logic sclk_delayed;
always_ff @ (posedge clk)
begin
  if (rst_n == '0)
    sclk_delayed <= '0;
  else
    sclk_delayed <= sclk;
end

always_ff @ (posedge clk or negedge rst_n)
begin
  if (rst_n == '0)
    begin
tx <= '0;
buff_update <= '0;
buff_latch <= '0;
tx_counter <= '0;
sofdone <= '0;
pardone <= '0;
par <= '0;
end
else if ( buff_update | pardone )
begin
  buff_update <= '0;
pardone <= '0;
end
else if ( buff_latch )
begin
  buff_latch <= '0;
end
else if ( sofdone == '1 )
sofdone <= '0;
else if ( init_tx )
begin
  buff_latch <= '1;
tx <= '1;
tx_counter <= {1'b0, first_byte_n};
  par <= (|first_byte_n) ? (~data) ^ 1'b1 : par;
end
else if ( sclk )
begin
  if ( tx_state == PAR)
  begin
    tx <= par;
tx_counter <= '0;
pardone <= '1;
end
else if ( tx_state == TX_ACTIVE )
begin
  tx <= tx_buff[0];

  par <= ~(|tx_counter) ? (~tx_buff) ^ 1'b1 : par;
  buff_update <= ~(&tx_counter[2:0]);
buff_latch <= ( &tx_counter[2:0] );

  tx_counter <= tx_counter + 4'h1;
end
else
begin
  tx_counter <= '0;
tx <= '0;
end
end
endmodule
3.4. The NFC Type 2 core

When all the modules above are working, i.e. the analog front end, the CRC generator and the RX/TX modules, the remaining part is the NFC type 2 core. This is the glue logic connecting the above modules, as well as deciding when to respond to the PCD requests, as well as what to respond. The flow diagram describing roughly the functionality of the NFC type 2 core is shown in fig. 11.

![The NFC type 2 core flow diagram](image-url)
As can be seen in fig. 7, the PICC enters the IDLE state when initiated. The PICC leaves this state when one of the ATQA and WUPA requests are received, which is answered with ATQA, and the PICC enters the RDY1 state. Moving to the RDY1 state is possible also when the PICC is in the HALT state and the WUPA request is received.

In the RDY1 and RDY2 states the PICC is listening for matching anti-collision frames. When a matching (but not necessarily finished) anti-collision frame is received, the PICC responds to this by finishing the anti-collision frame if it is not finished, and by sending the SAK value if the frame is finished and the CRC is ok.

The ACTIVE state is for the actual data transmission, and the PICC is listening for the READ and WRITE requests. If the READ request is received, the PICC reads 4 blocks of data beginning at the given address, and responds with these bytes. If a problem with reading the data (e.g. bad address), or the CRC was not matching, a NAK is received. If a WRITE request is received, the PICC writes one block of data into the given address. This is answered with an ACK if the operation was successful, and NAK otherwise.

What is not shown in this flow diagram is the shortcut, where the PICC can move directly from one of the RDY states into the ACTIVE state if a READ request addressing block 0 is received. If such a request is received, the PICC responds with the UID bytes, the BCC bytes, the LOCK bytes and the CAPABILITY CONTAINER bytes. These bytes are the bytes in the 4 first blocks, as can be seen in fig. 8.
4. TESTING THE PICC FUNCTIONALITY

Measurements of the analog front end can be done by measuring the voltage at the data-in pin on the FPGA to see if the demodulator is working, and by measuring the induced voltage over a coil in the field to see if the load modulator is working. These measurements were done during the REQA and ATQA request and response, and the results of these measurements can be seen in figures 12 and 13. In figure 13, the communication in both directions are shown because the load modulator modulates the signal at the input of the envelope detector. Also a closer look at how the load modulation signal looks in the field is shown in figure 14.

Figure 12: The NFC signal signal measured as the induced voltage over a coil in the NFC field.

Figure 13: The demodulated NFC signal at the input of the FPGA.
Figure 14: One bit of the NFC load modulation signal measured as the induced voltage over a coil in the NFC field.

This PICC circuit should work exactly the same way as a commercial PICC. The way to confirm its functionality is to use available NFC applications on an Android smart phone, which for example can be done in the following three steps. First, the PICC read functionality can be confirmed with the TagInfo application by NXP. Next, the write functionality can be confirmed with the TagWriter application by NXP, where the application also read the PICC after the write operation, to confirm that the write operation is succeeded. The last step is to write a message (“Hello World”) using the NFC data exchange format (NDEF), which should pop up on the android phone when read without an NFC application opened. All three steps were completed and the PICC showed the behavior it should. The results of the tests are shown in table 1, and pictures showing the successful read, write and NDEF operation of the PICC with an Android phone is shown in figures 15, 16 and 17. Larger versions of these pictures are also attached as attachments 1-3.

Table 1: Overview of the tests and whether the PICC passed or failed.

<table>
<thead>
<tr>
<th>Test</th>
<th>Pass/Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read by TagInfo</td>
<td>Pass</td>
</tr>
<tr>
<td>Write by TagWriter</td>
<td>Pass</td>
</tr>
<tr>
<td>NDEF pop-up in Android</td>
<td>Pass</td>
</tr>
</tbody>
</table>
5. DISCUSSION

In this text, the technical description of NFC-A type 2 PICCs were discussed, as well as the FPGA implementation of and NFC-A type 2 PICC. Also the circuit used when developing the PICC were briefly introduced. Code showing
how the PCD to PICC signal demodulation technique as well as the PICC to PCD modulation technique was shown. Also the flow diagram showing which responses should be sent and when they should be sent was shown. At last, the functionality of the PICC was evaluated. The analog front end was evaluated using an oscilloscope, and the functionality of the NFC Type 2 PICC core was evaluated using an Android phone featuring NFC.

Compared to another NFC type 2 PICC, the NXP Mifare Ultralight series, the functionality of the PICC was indeed the same. The difference however is the flexibility in the PICC, i.e. how the PICC can be developed. The PICC embedded in the FPGA can easily be extended to implement any kind of sensor interfaces or communication protocols. Further, by defining non-standard requests and answers, the PICC can be used as a bridge between NFC and another communication protocol, e.g. I2C. Also, the PICC can be altered to use a non-standard modulation technique to e.g. increase the speed of the communication link, or to increase the energy harvest possibilities.

Indeed this design was developed as a part of a research project, exploring the possibilities to communicate with sensors through NFC, for example by using a smartphone.
6. SUMMARY

The requirements of an NFC Forum PICC device have been discussed in this text, and how to implement an NFC Type 2 PICC on an FPGA. The implementation was done in SystemVerilog. Every step of the process of designing the NFC Type 2 PICC is shown, and the SystemVerilog code is presented for the important functions of receiving and sending data.

Further, the functionality of the PICC when programmed into a custom FPGA based NFC development platform was evaluated. The custom FPGA based NFC development platform was also discussed in this text. In the evaluation, waveforms at important nodes were shown, as well as the top level functionality of basic read and write routines.
REFERENCES


ATTACHMENTS

Attachment 1: Picture showing the results of the test when reading the PICC with the NXP TagInfo application.
Attachment 2: Picture showing the results of the test when writing to the PICC with the NXP TagWriter application.
Attachment 3: Picture showing the results of the test when reading the PICC with the stock Android NFC NDEF reader application.