# Debugging SPI between Teensy 3.5 and RFM95 using the RadioHead library

| | |
|---|---|
| Status: | Draft |
| Author: | W.J. (Pim) Niessen |
| Date: | 2017-07-02 |
| File name: | Teensy 3.5 and Semtech SX1276 debug PA2PIM V0.3.docx |

## Revision History

| Version | Date | Author | Changes |
|---|---|---|---|
| 0.2 | 2017-07-02 | W.J. (Pim) Niessen | First draft |
| 0.3 | 2017-02-03 | W.J. (Pim) Niessen | 2nd draft |
| | | | |

# 1. Debugging SPI between Teensy 3.5 and RFM95

## 1.1. Introduction

RFM95 modules based on Semtech LORA chips together with a Teensy 3.5 using the RadioHead library V1.77 require that the `TxDone` flag in the `RegIrqFlags` register (0X12) gets cleared twice before the flag is reset. This twice clearing of the TxDone flag was put in as a workaround for the infrequent lockup of software on various Teensy and RF modules supported by the library.

Lockups not only get reported with RFM95 modules but also with other LORA modules based on the Semtech SX1276/77/78 and79 chips.

This document describes further investigation as to why this workaround is needed.

## 1.2. Summary

The Serial Peripheral Interface (SPI) was suspected to be the root cause of the original lock-up problem. The Arduino IDE allows the Teensy 3.5 to be operated at clock rates at a multiple of 24 MHz. Up to 48 MHz no problems with lockups were encountered. At 72 MHz executing the main loop became unstable. That is every 5-7 iterations it would take longer than the standard 1 s through the loop.
At 96 and 120 MHz the software would lock up on the first iteration through the main loop.

It was found that at clock rates of 72 MHz and higher for the Teensy 3.5 the tnhold parameter of the NSS (Not System Select) signal from the SPI interface is not met.
The NSS signal becomes high too early in relation to the falling edge of the SCK (Clock) signal. This is not in line with the specification of the Semtech chips.

A temporary workaround by inserting a 1 µs delay in the actual SPI write and read functions removes the need for the double clearing of the `TxDone` flag. What's more: it is also a solid explanation for the observed lock up issues. A more elegant solution then inserting wait functions might be available in the future.

# 2. Data sheet, Semtech SX1276/77/78/79

LORA modules are available under various names. They are just about, if not all, based on the Semtech SX1276, SX1277, SX1278 or SX1279 chips. For the datasheet of these chips see Reference 6.2.

SPI information is located at two places in the datasheet but the information is sparse. The standard diagrams with signals and minimum time parameters are missing.

- **2.5.6 Electrical Specifications,**
  Table 11

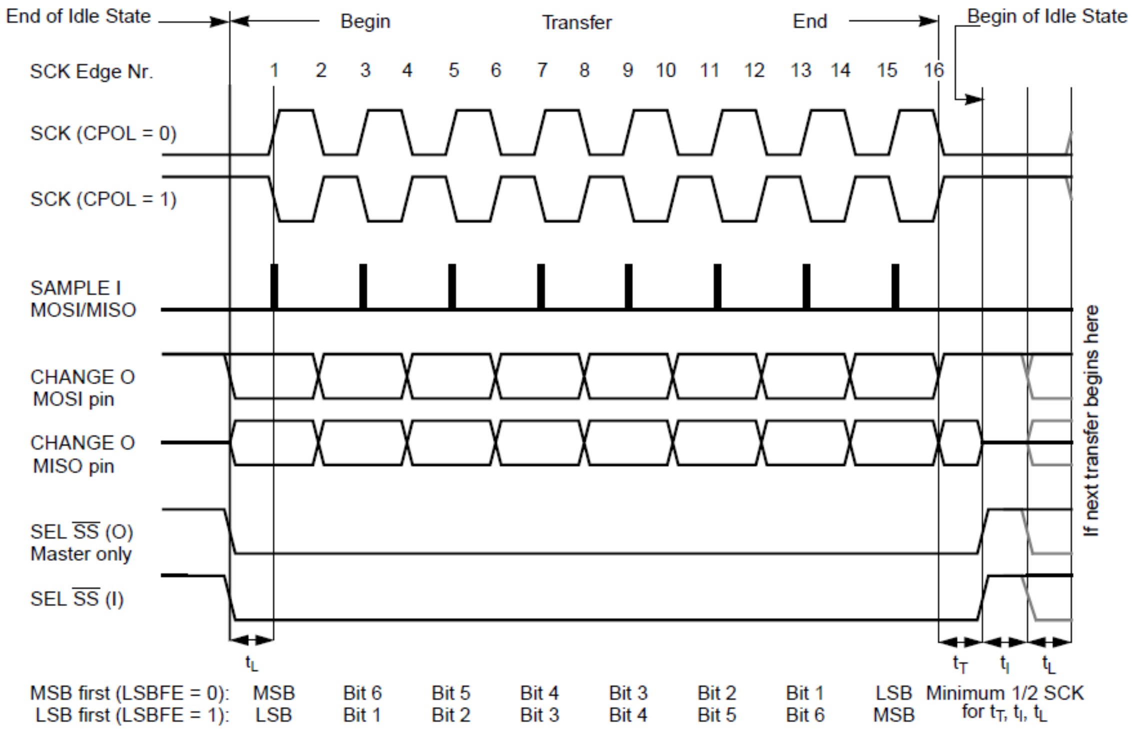
| Symbol | Description | Conditions | Min | Unit |
|---|---|---|---|---|
| Tnhold | From **SCK falling** edge to **NSS rising** edge, normal mode | - | 100 | ns |

  - "normal" mode is not specified in relation to the SPI timing.
  - The meaning of T_DATA parameter of min. 250ns at the bottom of the table is not specified.

- **4.3. SPI Interface**
  - SPI for the Semtech chips operates in Mode 0. (CPOL = 0 and CPHA = 0)
  - An address byte is formed with the MSB is 1 for write access and 0 for read access.
  - Then 7 bits of the address, MSB first.

# 3. SPI specification

A SPI specification from Motorola/Freescale, the ïnventors" of the SPI is found at Reference 6.1. Tnhold in the Semtech specification is $t_T$ in the diagram below taken from Reference 6.1


End of Idle State
Begin
Transfer
End
Begin of Idle State
SCK Edge Nr.
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
SCK (CPOL = 0)
SCK (CPOL = 1)
SAMPLE I MOSI/MISO
CHANGE O MOSI pin
CHANGE O MISO pin
SEL $\overline{SS}$ (O) Master only
SEL $\overline{SS}$ (I)
If next transfer begins here
$t_L$
$t_T$
$t_I$
$t_L$
MSB first (LSBFE = 0): MSB Bit 6 Bit 5 Bit 4 Bit 3 Bit 2 Bit 1 LSB
LSB first (LSBFE = 1): LSB Bit 1 Bit 2 Bit 3 Bit 4 Bit 5 Bit 6 MSB
Minimum 1/2 SCK for $t_T$, $t_I$, $t_L$

$t_L$ = Minimum leading time before the first SCK edge

$t_T$ = Minimum trailing time after the last SCK edge

$t_I$ = Minimum idling time between transfers (minimum $\overline{SS}$ high time)

$t_L$, $t_T$, and $t_I$ are guaranteed for the master mode and required for the slave mode.

**Figure 4-2 SPI Clock Format 0 (CPHA = 0)**

*Figure 1. SPI timing acc. to Motorola/Freescale specification*

# 4. Timing diagrams.

Timing diagrams were collected with a DSO for the Teensy 3.5 operating at the various clock frequencies. Light blue is the SCK signal. In dark blue is the NSS signal of the SPI to the RFM95 module.

## 4.1. Teensy 3.5 @ 120 MHz

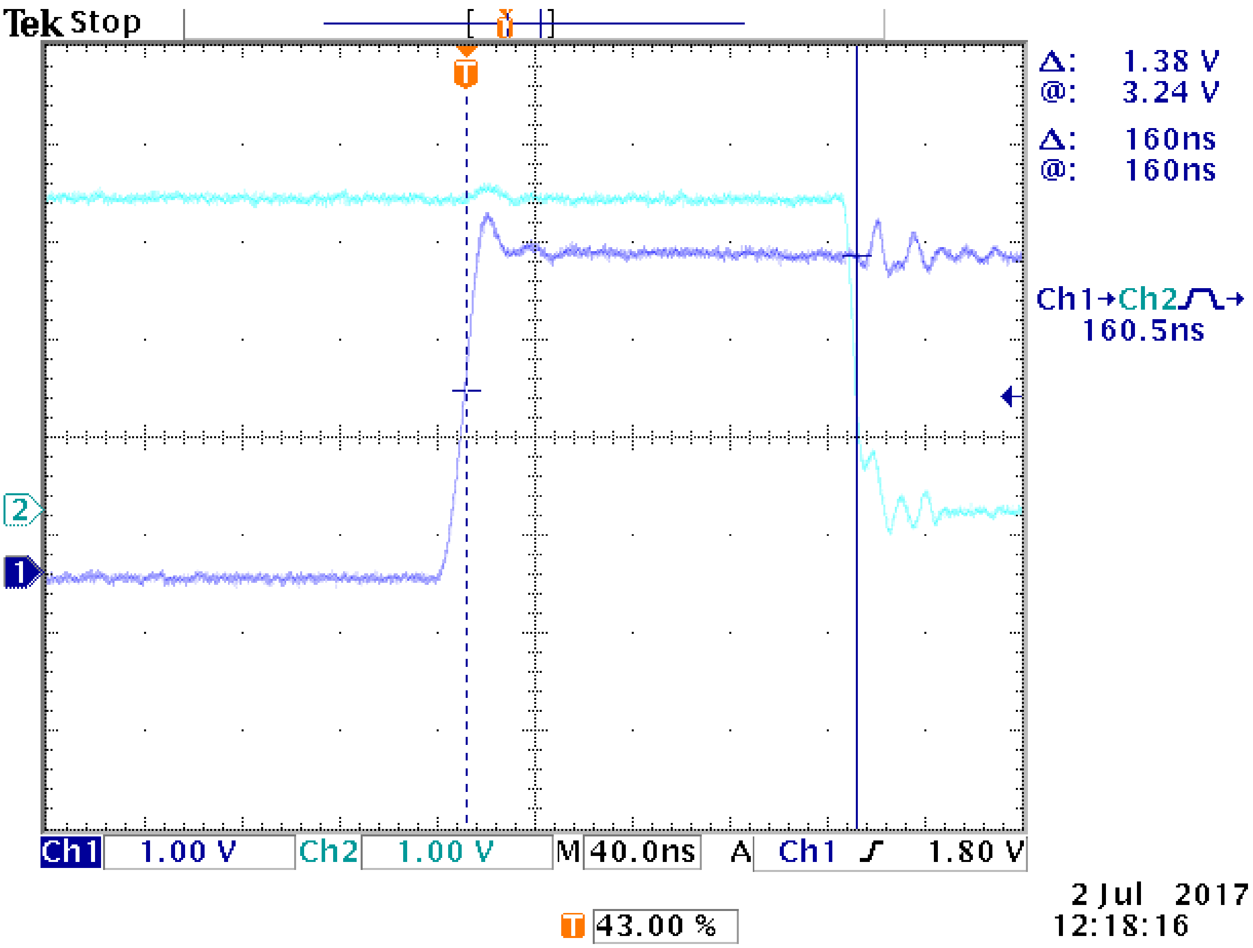


*Figure 2. Teensy 3.5, 120 MHz, Fastest, NSS (Dark BLue), SCK (Light BLue)*

### 4.1.1. Conclusion

Error condition. NSS becomes high 160ns before SCK is low. It should stay low at least 100ns <u>after</u> SCK is low.

NSS becomes high 260ns too early.

## 4.2. Teensy 3.5 @ 96MHz

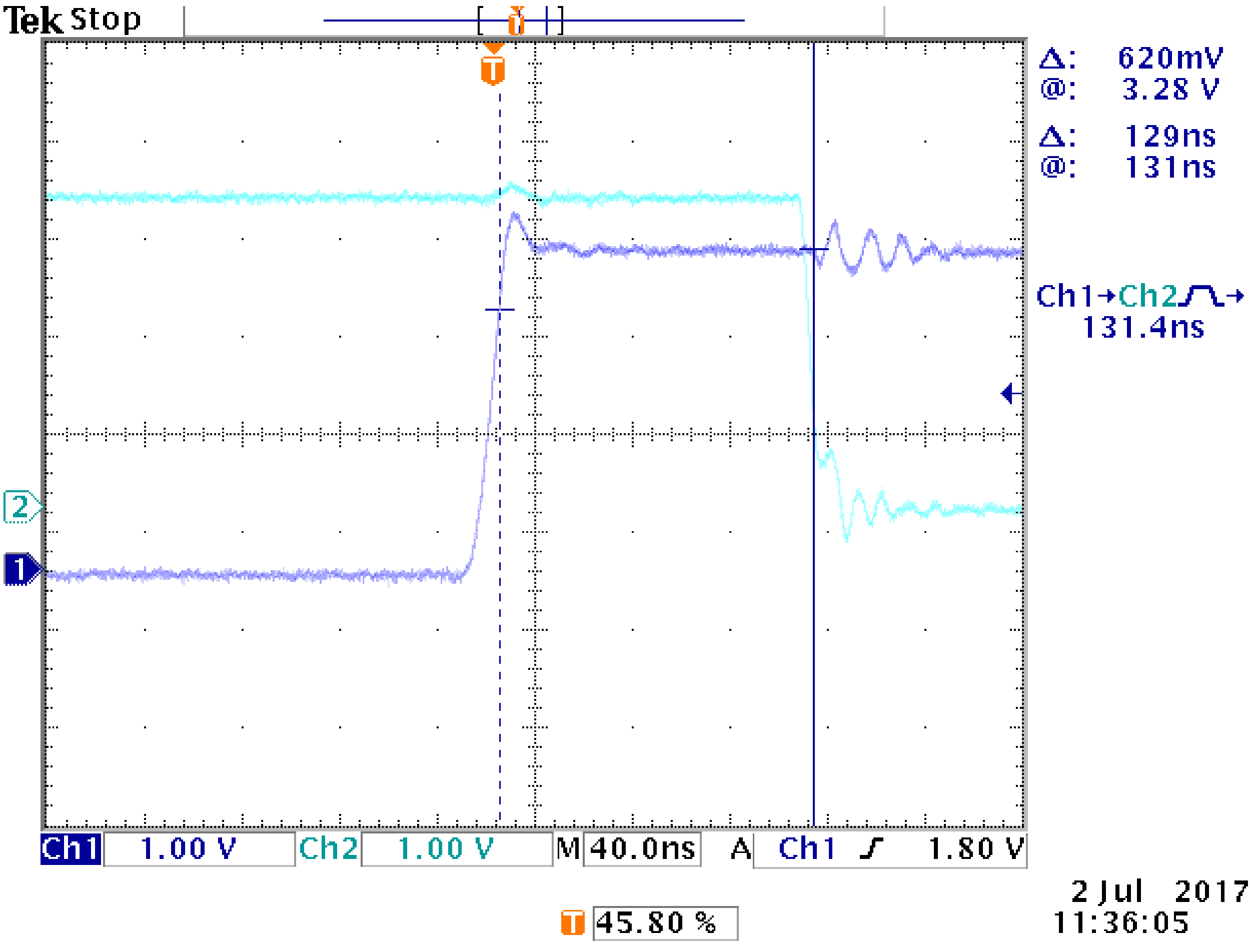


*Figure 3. Teensy 3.5, 96 MHz, Fastest, NSS (Dark BLue), SCK (Light BLue)*

### 4.2.1. Conclusion

Error condition. NSS becomes high 130 ns before SCK is low. It should stay low at least 100ns <u>after</u> SCK is low.

NSS becomes high 230 ns too early.

## 4.3. Teensy 3.5 @ 72 MHz

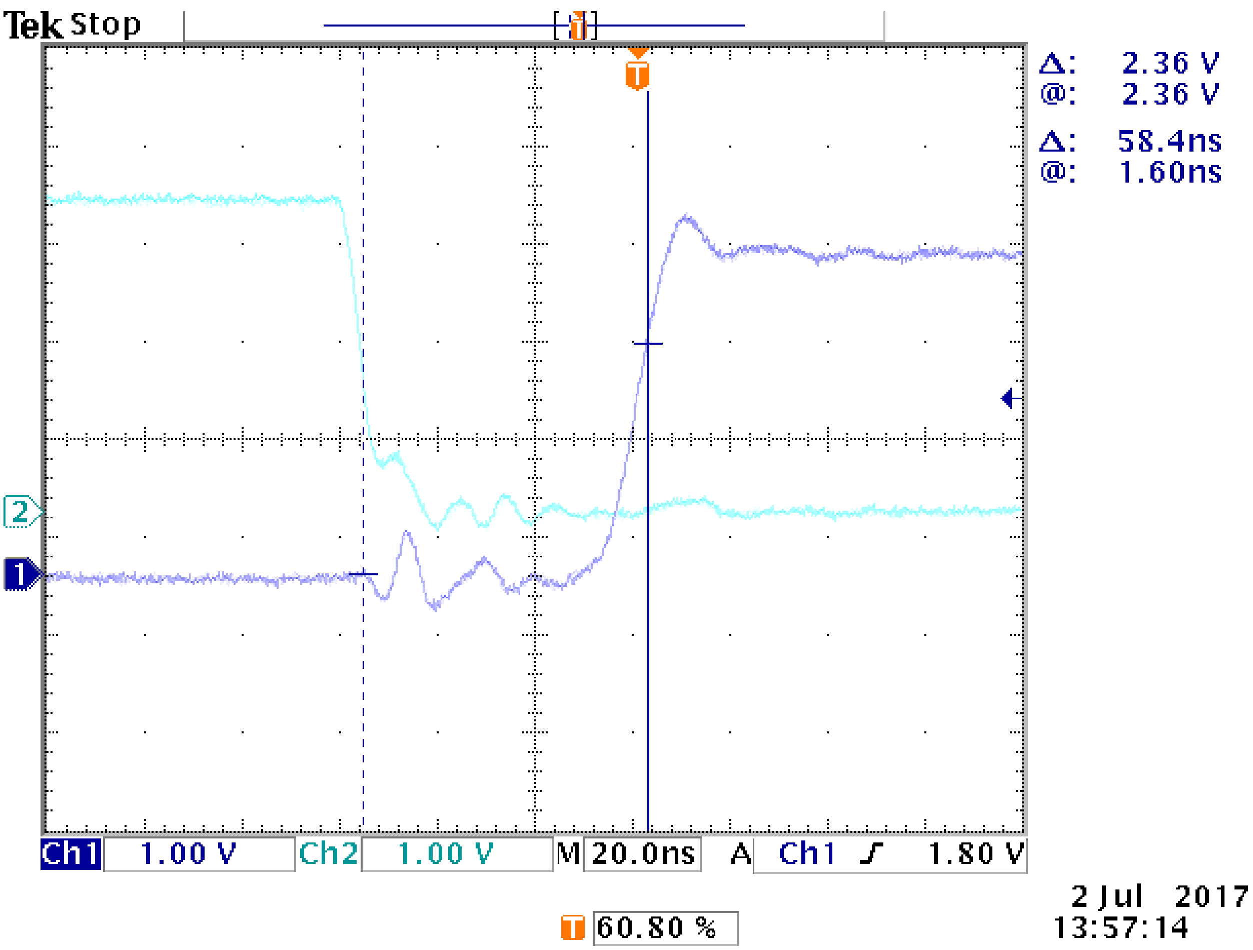


*Figure 4. Teensy 3.5, 72 MHz, Fastest, NSS (Dark BLue), SCK (Light BLue)*

### 4.3.1. Conclusion

Error condition. NSS becomes high 60 ns after SCK is low. It should stay low at least 100ns <u>after</u> SCK is low.

NSS becomes high 40 ns too early.

## 4.4. Teensy 3.5 @ 48 MHz

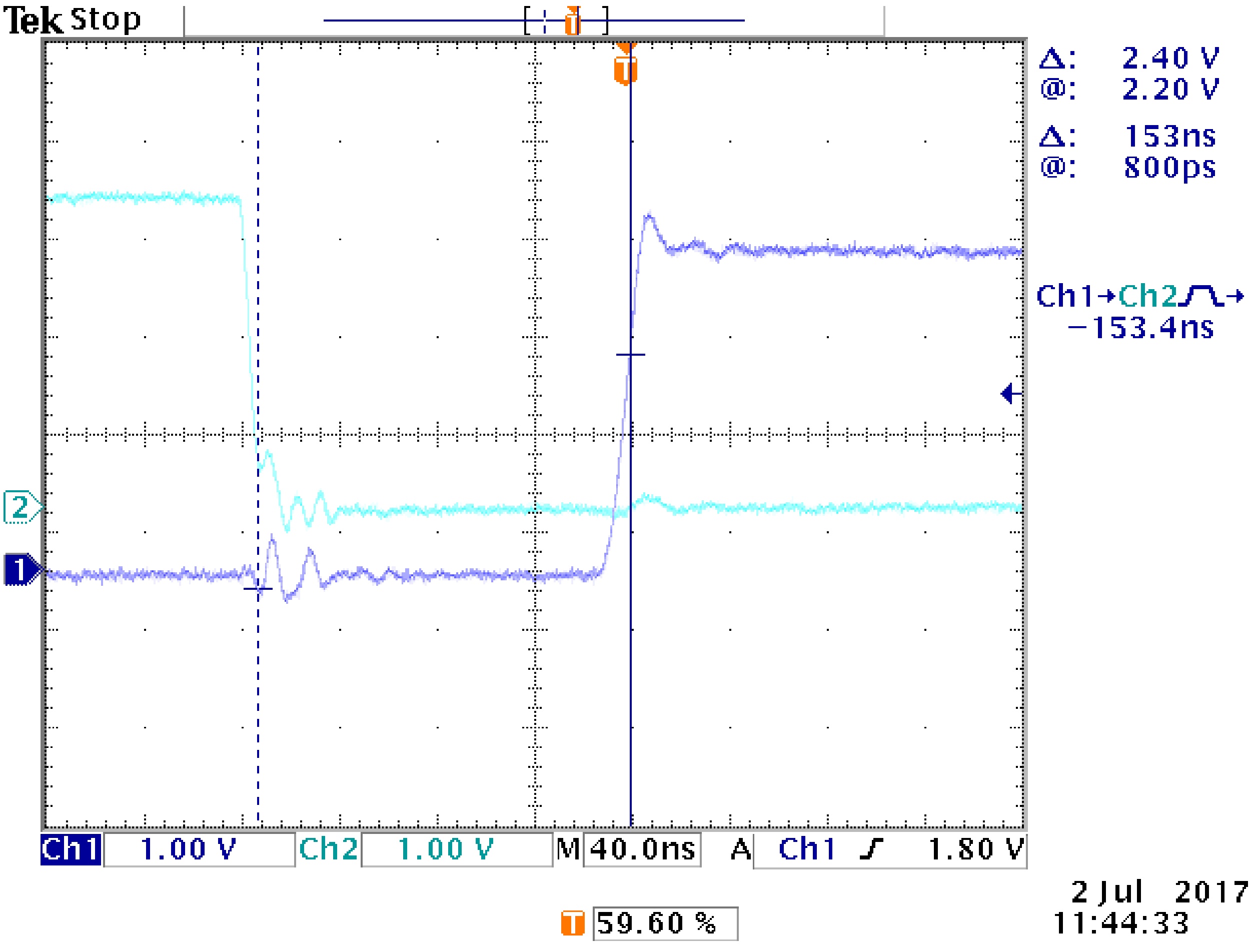


*Figure 5. Teensy 3.5, 48 MHz, Fastest, NSS (Dark BLue), SCK (Light BLue)*

### 4.4.1. Conclusion

Normal Condition. NSS becomes high 150 ns after SCK is low.

Specification is that it should stay low at least 100 ns after SCK is low as well so this condition is met.

## 4.5. Teensy 3.5 @ 24 MHz

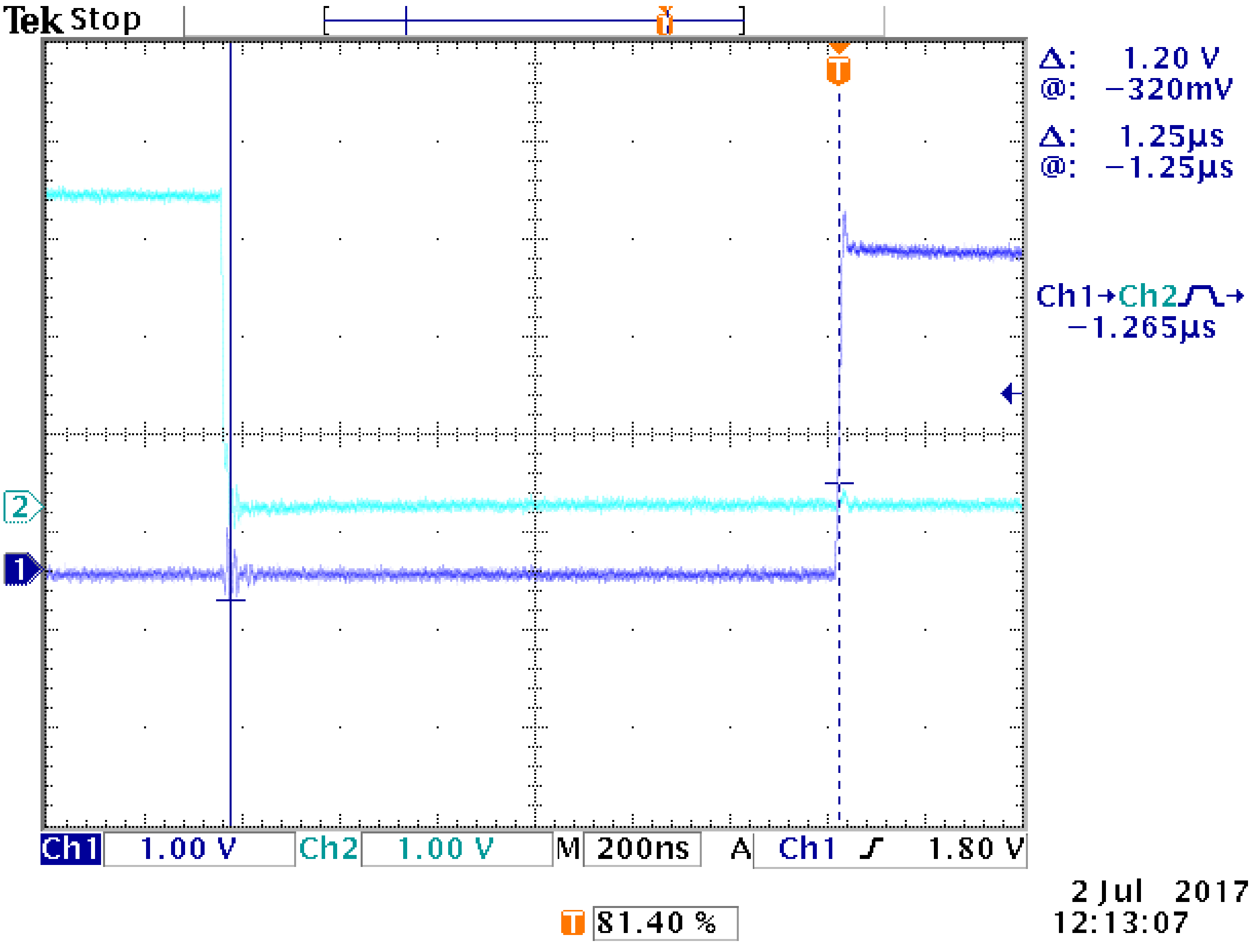


*Figure 6. Teensy 3.5, 24 MHz, Fastest, NSS (Dark BLue), SCK (Light BLue)*

### 4.5.1. Conclusion

Normal Condition. NSS becomes high 1260 ns after SCK is low.

Specification is that it should stay low at least 100 ns after SCK is low as well so this condition is met.

### 4.6. Timing measurements summary.

| **Teensy 3.5 clock frequency (MHz)** | **Condition** | **Delay between Rising edge NSS and falling edge of last SCK pulse (ns)** | **Timing margin with respect to tnhold parameter (ns)** |
|---|---|---|---|
| 120 | Error | -160 | -260 |
| 96 | Error | -130 | -230 |
| 72 | Error | 60 | -40 |
| 48 | Normal | 150 | 50 |
| 24 | Normal | 1260 | 1160 |

## 5. Conclusion

The timing of the SPI interface between the Teensy 3.5 and the RFM95 module using the RadioHead library is only meeting the SPI specifications for the NSS and SCK signals when operating at 24 MHz or 48 MHz. At 72 MHz the timing margin is not adhered to. At 96 MHz and 120 MHz NSS becomes high before SCK has gone low. These timing issues match with the observed lockups of the Arduino sketch from 7.2 when it is compiled for the various clock frequencies.

An insertion of a 1µs delay in the

```
spiRead
spiWrite
spiBurstRead
spiBursWrite
```

functions in the RHSPIDriver.cpp code prior to raising the NSS line removes these timing issues. See the code snippet below:

```
uint8_t RHSPIDriver::spiWrite(uint8_t reg, uint8_t val)
{
    uint8_t status = 0;
    ATOMIC_BLOCK_START;
    digitalWrite(_slaveSelectPin, LOW);
    status = _spi.transfer(reg | RH_SPI_WRITE_MASK); // Send the address
with the write mask on
    _spi.transfer(val); // New value follows
   //debug start, go watching paint dry
   delayMicroseconds(1);
   //debug end, it's dry
    digitalWrite(_slaveSelectPin, HIGH);
    ATOMIC_BLOCK_END;
    return status;
```

The double clearing of the `TxDone` flag in the interrupt handling code in RH_RF95.cpp

```
spiWrite(RH_RF95_REG_12_IRQ_FLAGS, 0xff); // Clear all IRQ flags
spiWrite(RH_RF95_REG_12_IRQ_FLAGS, 0xff); // Clear all IRQ flags
```

is no longer needed and one of these lines can be removed.

# 6. Reference

6.1. Motorola Freescale SPI specification:
http://read.pudn.com/downloads67/doc/240308/SPI%20SPEC/S12SPIV4.pdf

6.2. Semtech datasheet:
http://www.semtech.com/images/datasheet/sx1276.pdf

# 7. Environment

## 7.1. Software

- Teensy 3.5
- Arduino IDE 1.8.1
- TeensyDuino 1.37
  - https://www.pjrc.com/teensy/td_137/TeensyduinoInstall.exe
  - Does not contain the workaround in RH-RF95.cpp of clearing the interrupt flag twice.
- RadioHead library 1.77
  - http://www.airspayce.com/mikem/arduino/RadioHead/RadioHead-1.77.zip
  - Contains a workaround by clearing the Interrupt flag twice in RH_RF95.cpp

```
 // Sigh: on some processors, for some unknown reason, doing this only once does
not actually
//clear the radio's interrupt flag. So we do it twice. Why?
    spiWrite(RH_RF95_REG_12_IRQ_FLAGS, 0xff); // Clear all IRQ flags
    spiWrite(RH_RF95_REG_12_IRQ_FLAGS, 0xff); // Clear all IRQ flags
```

## 7.2. Sketch

Below is the software code for the Arduino IDE used for testing the lockup phenomena and the workaround. It is a minimized version of the example software included with the RadioHead library.

```
// rf95_client.pde
#include <SPI.h>
#include <RH_RF95.h>

RH_RF95 rf95(10,2); // Teensy 3.5
uint16_t iterations;

void setup()
{
  Serial.begin(9600);
  while (!Serial) ; // Wait for serial port to be available
  if (!rf95.init())
    Serial.println("init failed");
  iterations = 1; //humans start at 1 :-)
}
```

```
void loop()
{
  Serial.print (iterations);
  Serial.println(" Sending to rf95_server");
  iterations++;
  uint8_t data[] = "Hello World!";
  rf95.send(data, sizeof(data));

}
```