

Chilipepper User’s Guide

|  |
| --- |
| Version 1.2  2/18/2014  Latest Board Rev. A2 |

[embedded@toyon.com](mailto:embedded@toyon.com)

[embedded@toyon.com](mailto:embedded@toyon.com)

[embedded@toyon.com](mailto:embedded@toyon.com)

Table of Contents

[1 Foreword 1](#_Toc380499428)

[2 Introduction 2](#_Toc380499429)

[3 Interface Signals 3](#_Toc380499430)

[3.1 MCU and RF-FE Signaling 4](#_Toc380499431)

[3.2 Transmit Signaling and Digital to Analog Control (DAC) Driver 5](#_Toc380499432)

[3.3 Receive Signaling and Analog to Digital Control (ADC) Driver 8](#_Toc380499433)

[3.4 Constraints 11](#_Toc380499434)

[4 Chilipepper DIP Switch Settings 12](#_Toc380499435)

[5 FPGA to MCU Serial Protocol 13](#_Toc380499436)

[6 Reflashing MCU Firmware 15](#_Toc380499437)

[7 Precautions 17](#_Toc380499438)

[8 Reference Designs 18](#_Toc380499439)

[9 References 19](#_Toc380499440)

[Appendix A FAQ A](#_Toc380499441)

[Appendix B Board Revisions B](#_Toc380499442)

[Appendix C What is a Chilipepper? C](#_Toc380499443)

[Appendix D Example Constraints D](#_Toc380499444)

# Foreword

When I was a senior in college in 1997 I was like any other engineer. I wanted to build something. I wanted to make something work. But, when I went looking for jobs I saw there were two groups. The first did simulations and the other implemented stuff. I wanted to do both. So, like anyone who is unhappy and doesn’t really know what to do with their life I went to graduate school.

Entering graduate school I was thrilled with the math and many of the labs were great. But, where were the radios? I thought, ok once I move on from this master’s stuff and start graduate work I’ll get to build something. But, besides some very minor catastrophic failures with hardware, I mostly published lots and lots of paper.

After graduating in 2003 and entering industry I thought ok this is it. I’m finally going to build something that works! But, I found the same great divide there was in 1997. There were systems engineers and there were hardware engineers. They talked. But, they didn’t talk very well. Furthermore, there was almost no way to build a working radio besides building completely custom radio frequency (RF) front ends. And, we began to do this culminating in several custom single and multiple-input multiple-output (MIMO) transceivers. But, boy were they expensive and inflexible.

Chilipepper and the general MATLAB to HDL framework presented is the culmination of over a decades worth of frustration and exploration. For the first time I feel there is a way for a single engineer, or small team, to build a functional wireless radio that has a straightforward path to a real product.

I hope you enjoy Chilipepper. And if you don’t let’s figure out how to make it better.

-rich cagley Oct. 2012

# Introduction

Chilipepper is a radio frequency front end (RF-FE) meant for rapid prototyping of physical layer waveforms. The board complies with the VITA 57 standard for FPGA Mezzanine Cards (FMC). The radio itself is built around a Lime Microsystems LMS6002 radio frequency integrated circuit (RF-IC). The LMS6002 is a highly integrated device with most major functional elements within the common package. This includes analog and digital data converters, both transmit and receive synthesizers, filtering, and multiple stages of amplification and gain control.

While the LMS6002 is a highly integrated device, it is still complicated, requiring many weeks of development and some relatively expensive laboratory equipment in order to make the device functional and operate well. As Chilipepper is targeted to those just getting started with wireless radio design, we have placed a microcontroller unit (MCU) onboard to handle calibration and configuration duties. The MCU also masks the register control of the LMS6002, which requires and nondisclosure agreement (NDA) to be in place with Lime Microsystems. This is something most users do not want to deal with during initial development.

# Interface Signals

In this section we will cover the interface and control signals that you will need to have present in your FPGA design. We suggest mapping MCU and RF-FE signaling to a microcontroller running on the FPGA. The TX/DAC and RX/ADC signals should be driven by FPGA logic.

The RF-FE is driven by a 40 MHz oscillator that is present on Chilipepper. This source drives the synthesizers and other circuitry on the RF-IC. While not mandatory, we suggest basing your baseband design around this clock, which is returned with the schematic signal name in Table 1.

Table 1 - Clocking.

|  |  |  |
| --- | --- | --- |
| Signal Name | FMC Pin Location | Function |
| PLL\_CLK\_OUT | CLK1\_P | Main clock from RF-RE to the FPGA. This is a 40 MHz signal. |

## MCU and RF-FE Signaling

There is an AVR microcontroller present on Chilipepper that is responsible for the bulk of calibration and control of the RF-FE. The relevant signals are given in Table 2. We suggest the following power-on procedure:

1. Disable all clocking and signaling for DAC and ADC lines (all signals in Sections 3.2 and 3.3)
2. Set TX\_EN and RX\_EN high (during operation these can be brought low to save power)
3. Bring MCU\_RESET high then low then high (this resets the MCU)
4. Wait for MCU\_INIT\_DONE to go high

Once the MCU has finished initialization you will also notice that there is a flashing light on Chilipepper indicating the RF-FE is calibrated. You may now turn the power amplifier on/off and change transmit and receive modes.

Table 2 - MCU interface signals.

|  |  |  |
| --- | --- | --- |
| Signal Name | FMC Pin Location | Function |
| MCU\_UART\_TX/RX | TX - LA10\_P  RX - LA06\_P | Serial interface from FPGA to Chilipepper’s AVR MCU. See Section 3 for details. |
| MCU\_INIT\_DONE | LA14\_P | Signal from MCU to FPGA indicating the MCU has finished calibration. Active high. |
| MCU\_RESET | LA14\_N | Signal from FPGA to MCU causing it to reset. Active low. |
| M1\_TR\_SW | LA18\_CC\_P | Transmit/receive switch control for RF-FE. Setting this signal low enables transmit and high enables receive. |
| M1\_PA\_EN | LA27\_P | RF-FE power amplifier control. Setting this signal low disables the PA and high enables PA. There is no control over power output levels through direct PA interaction and instead this should be done through the serial port interface. |
| TX/RX\_EN | TX - LA25\_P  RX - LA25\_N | Set these high to enable to TX/RX portions of the RF-FE. The MCU will not begin calibration until these signals are set high |

## Transmit Signaling and Digital to Analog Control (DAC) Driver

TX/DAC control is straightforward with signals described in Table 3. A timing diagram is shown in Figure 2. Note that the TX\_CLK signal should be twice the TX\_IQ\_SEL signal frequency. Inphase (I) and quadrature (Q) data is interleaved with I corresponding to TX\_IQ\_SEL being low and Q corresponding to TX\_IQ\_SEL being high.

Table 3 – TX/DAC interface signals.

|  |  |  |
| --- | --- | --- |
| Signal Name | FMC Pin Location | Function |
| TX\_CLK | LA29\_P | Falling edge latching clock from the FPGA to the DAC. Can be driven up to 40 MHz and should be twice the sample rate of the TX data lines |
| TX\_IQ\_SEL | LA31\_P | In order to conserve pins, inphase and quadrature signals on transmit are interleaved. This binary signal selects whether I or Q is latched into the DAC during the rising edge of TX\_CLK. |
| TXD[0:11] | 0 - LA24\_P; 1 - LA24\_N  2 - LA21\_N; 3 - LA20\_N  4 - LA22\_N; 5 - LA19\_P  6 - LA21\_P; 7 - LA19\_N  8 - LA22\_P; 9 - LA28\_P  10 - LA28\_N; 11 - LA29\_N | There are 12 bits going to the DAC. TXD0 is the least significant bit and TXD11 is the most significant bit. Data is in 2’s complement format. |

**TX Clocks**

A list of the clocks used for transmitting as well as their explanations is given below.

1. **PLL Clock -** This clock signal is sent from the Chilipepper to the FPGA. It’s the clock used to derive the other clock signals for the transmit core. It has a frequency of 40 MHz, and is constant regardless of the design it is used for.

Building on this PLL clock input, there are 4 other important clock signals used within the design.

1. **RX Clock –** This clock is sent from the FPGA to the physical ADC on the Chilipepper. For the Chilipepper Labs, this clock should be 40 MHz, however for other designs the frequency of this clock may vary. Additionally, this clock can have arbitrary phase, as no other signals depend on the phase of this clock; due mostly to the presence of the RX return clock which will be mentioned later.
2. **TX Core Clock –** This clock is used to drive the TX core logic, which is responsible for creating the I and Q channel TX data signals. The clock is not sent to any other devices, and is thus an FPGA internal clock. By default, the FPGA is **rising edge triggered** and causes the TX Data to change on the rising edge as shown in Figure 1 below. Additionally, this clock has a frequency of 20 MHz, which is half the frequency of the other 3 PLL derived clocks. This is due to the data interleaving required by the Chilipepper DAC.
3. **DAC Driver Clock** **–**This clock is used to drive the Interleaving logic. The DAC Driver receives data from the TX core and places it in a single TXD signal to be sent to the Chilipepper DAC. Therefore this clock must have the same phase offset as the TX core clock.
4. **TX clock –** This clock is sent from the FPGA to the physical DAC on the Chilipepper. Like the RX clock, the TX clock should also be 40 MHz for the Chilipepper Labs, yet for other designs may vary. Since this clock will be used to sample data sent to the Chilipepper DAC, the phase of this clock **relative to the TX data** is important. As mentioned earlier, the DAC Driver clock is rising edge triggered, which means new TX data will become available on the rising edge of the DAC Driver Clock. However, since the physical Chilipepper DAC also samples data on the rising edge of the TX clock, if the TX and DAC Driver clocks have the same rising edge (same phase), we will run into the issue of data being latched in the DAC at the same time it changes in the DAC Driver logic. This can lead to unstable data communication between the devices. Therefore, **the TX clock should have a phase offset of 180** to overcome this.

Using Figures 1 and 2, you can see the requirements for the relative phases of each signal in the TX chain. Figure 2 shows the theoretical requirements while Figure 1 shows a real time plot of these signals using ChipScope.

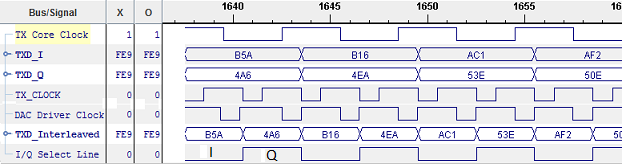


Figure 1: Shows the relationship between the TX Core clock, and the TX Clock, TX Data, and I/Q Select Line sent to the DAC

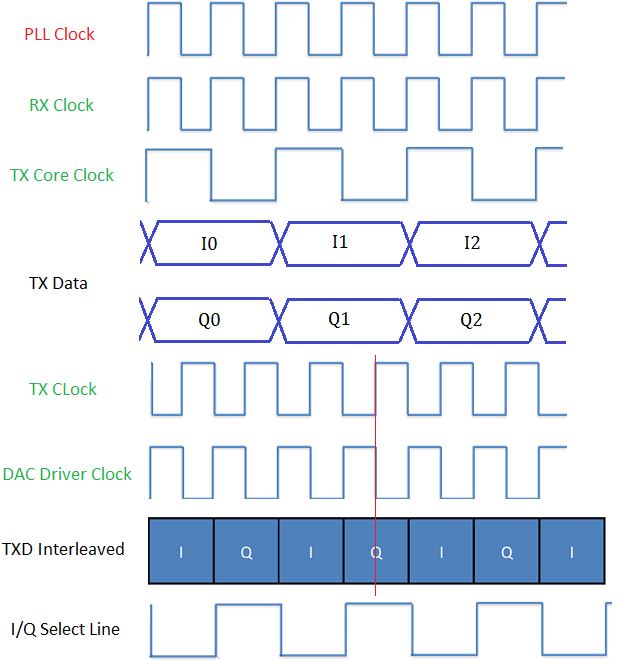
****

Figure 2: Clock signals used for transmit. As shown above (red line), by shifting the TX Clock signal by 180, you ensure that the TXD Data is stable during the rising edge of the TX Clock sent to Chilipepper.

## Receive Signaling and Analog to Digital Control (ADC) Driver

Receive Signaling and Analog to Digital Control (ADC) Driver

RX/ADC signaling is very similar to the TX/DAC.

Table 4: RX/ADC interface signals

|  |  |  |
| --- | --- | --- |
| Signal Name | FMC Pin Location | Function |
| RX\_CLK | LA05\_P | Rising edge latching clock from the FPGA to the ADC. Can be driven up to 40 MHz and should be twice the sample rate of the RX data lines. Note this clock should not be the same clock that drives your receive processing as it is not guaranteed to align with the RX\_IQ\_SEL and RXD signals. |
| RX\_CLK\_RET | CLK0\_P(Rev. 1 and 2)  LA16\_N(Rev. 0) | This clock is from Chilipepper to the FPGA. It can be used to align the returned RX\_IQ\_SEL and RXD signals with a clock. Both the ADC and the FPGA sample data on the rising edge and as such the user should skew RX\_CLK\_RET to have a phase offset of +90 degrees before using this clock to process receive data. |
| RX\_IQ\_SEL | LA01\_CC\_P | In order to conserve pins, inphase and quadrature signals on receive are interleaved. This binary signal selects whether I or Q is currently output from the ADC. It is sent from Chilipepper to the FPGA. |
| RXD | 0 - LA04\_P; 1 - LA08\_P  2 - LA04\_N; 3 - LA08\_N  4 - LA07\_P; 5 - LA12\_P  6 - LA07\_N; 7 - LA11\_P  8 - LA16\_P; 9 - LA12\_N  10 - LA11\_N; 11 - LA15\_P | There are 12 bits going to the ADC. RXD0 is the least significant bit and RXD11 is the most significant bit. Data is in 2’s complement format. |

**RX Clocks**

When using the RX Return clock to design your baseband receiver procesing, it is reccomended that you follow the following clocking procedures. Like the PLL clock, there is a clock used to derive RX clock signals called the RX Return Clock.

1. **RX Return Clock -** This clock is specifically aligned with the RXD input from the ADC to allow for proper data sampling. This alignment specifically means the data **changes on the rising edge of the RX Return clock**.

Building on this RX Return Clock, there are 2 other important clock signals used within the design.

1. **ADC Driver Clock** **–**This clock is used to drive the deinterleaving logic. The ADC Driver receives RX data from the Chilipepper ADC, and splits the data into I and Q channels before sending it to the RX core for processing. By default, the FPGA is **rising edge triggered** and samples this RX data on the rising edge of the ADC Driver Clock. Due to the fact that during the rising edge of the RX Return Clock, data is changing and unstable, this clock **must also have an offset of 180** applied to it. This ultimately leads to the timing diagram shown in Figure 4 below.
2. **RX Core Clock –** This clock is used to drive the RX core logic, which is responsible for processing the I and Q channel RX data signals. Additionally, this clock has a frequency of 20 MHz, which is half the frequency of both the RX Return Clock and the ADC Driver clock. This is due to the data deinterleaving performed by the ADC driver. Since this core receives its input directly from the ADC Driver core, its phase must also be offset 180 to stay consistent with the ADC Driver input within the FPGA.

Using Figures 3 and 4, you can see the requirements for the relative phases of each signal in the RX chain. Figure 3 shows the theoretical requirements while Figure 4 shows a real time plot of these signals using ChipScope.

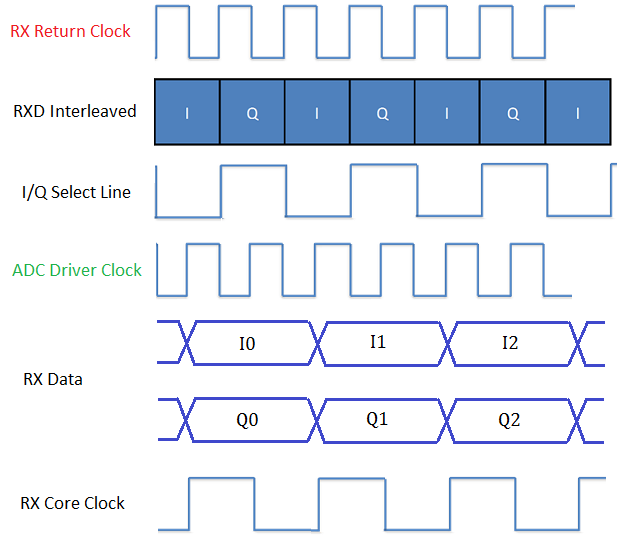
****

Figure 3: Clock signals used for receive

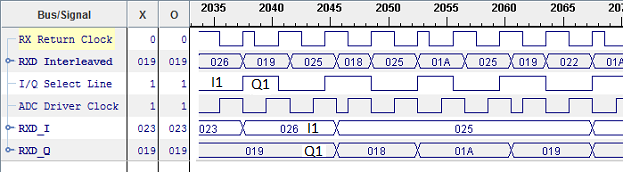


Figure 4: Shows the relationship between the ADC Driver clock, and the RX Return Clock, RX Data, and I/Q Select Line received from the ADC.

## Constraints

Ideally users will probe signaling from the FPGA to Chilipepper to determine drive strength and any overshoot. On the ZED board with 2.5V signaling we used FAST switching and the lowest possible drive strengths of 4 for clocking. Please refer to Appendix C.

# Chilipepper DIP Switch Settings

DIP switches on the board control transmit and receive modes. \*Note, only mode 1 is currently functional. Please inquire if other modes are desired.

1. TDD in 2.4 GHz band
2. FDD in 2.2 GHZ band
3. Wideband receive only

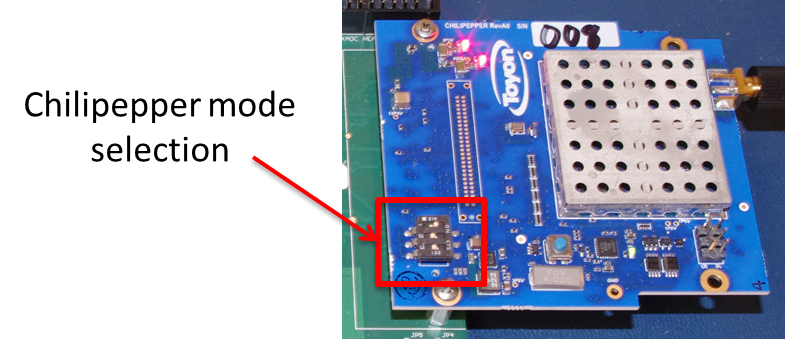


Figure 5 - Illustrating default state of DIP switches.

# FPGA to MCU Serial Protocol

A simple four byte packet is used to configure and control Chilipepper. The FPGA must follow the format illustrated in Figure 6. Use 9600 baud with no parity checking or hardware flow control.

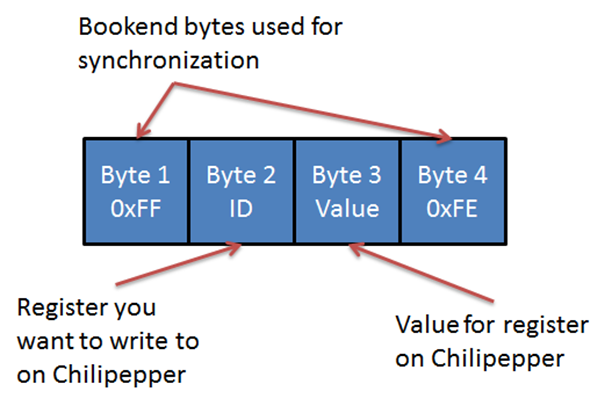


Figure 6 - Serial packet format for Rx/Tx with Chilipepper.

Tables 5 and 6 provide information on the control that is made available to the FPGA. Note that Chilipepper will not initiate transmit packets back to the FPGA; these will take place only in response to a packet sent from the FPGA to Chilipepper. The FPGA must read the four return bytes every time a packet is sent.

Table 5 - Serial packet transmission from FPGA to Chilipepper.

|  |  |  |  |
| --- | --- | --- | --- |
| Register ID | Value range | Default | Functional description |
| 0x00 | 0 or 1 | N.A. | Controls LED on Chilipepper. 0 is off and 1 is on. |
| 0x01 | 0 to 25 | 0 | Controls Tx gain. Value is in dB. |
| 0x02 | 0 to 57 | 14 | Controls Rx gain. Value is in dB. Note that values above 30 have resolution of 3. Hence, 30, 31, 32 all generate equivalent Rx gains. |
| 0x03 | 2.4 to 2.485 GHz (mode 1)  1.9224 to 1.9776 (mode 2)  0.3 to 3 GHz (mode 3) | 2.4 GHz  1.9224 GHz  2.4 GHz | Controls the TX center frequency. The value should be in units of Kilohertz and requires three Hexadecimal data bytes to fully specify the frequency. |
| 0x04 | 2.4 to 2.485 GHz (mode 1)  2.11 to 2.17 GHz (mode 2)  0.3 to 3 GHz (mode 3) | 2.4GHz  2.11 GHz  2.4 GHz | Controls the RX center frequency. The value should be in units of Kilohertz and requires three Hexadecimal data bytes to fully specify the frequency. |
| 0x05 | 0.75 to 14 MHz | 14 MHz | Controls the TX LPF bandwidth. The supported filter cutoff frequencies can be found below1. If a bandwidth within this range is specified that is not one of the supported cutoff frequencies, the next highest bandwidth will be used. Any specified bandwidth higher than the max will be ignored. |
| 0x06 | 0.75 to 14 MHz | 14 MHz | Controls the RX LPF bandwidth. The supported filter cutoff frequencies can be found in below[[1]](#footnote-1). If a bandwidth within this range is specified that is not one of the supported cutoff frequencies, the next highest bandwidth will be used. Any specified bandwidth higher than the max will be ignored. |
| 0x07 | 0 or 1 | N.A. | Increment or decrement the TX center frequency by 100 KHz. 0 – increment 1 – decrement. |
| 0x08 | 0 or 1 | N.A. | Increment or decrement the RX center frequency by 100 KHz  0 – increment 1 – decrement. |

\*Note changing the center frequency either by value or incrementally in mode 1 or mode 3 will change both the TX and RX frequencies simultaneously.

Table 6 – Return serial packet transmission from Chilipepper to FPGA.

|  |  |  |
| --- | --- | --- |
| Register ID | Value range | Functional description |
| 0x00 | 0 or 1 | Query Chilipepper status. 0 – indicates problem. 1 – everything operating normal. |
| 0xYZ | NA | Any other ID/value combination sent from FPGA is simply returned. |

# 

# Reflashing MCU Firmware

You will need to install Atmel Studio 6.0 along with any service packs. A screenshot of installed tools is shown in Figure 7.

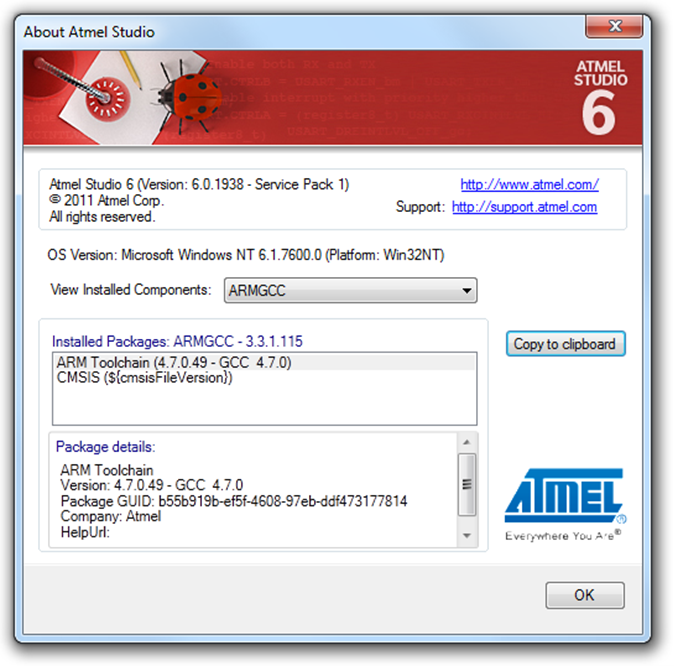


Figure 7 - Current version used in development.

You will also need to install the driver for the programmer and optionally update the firmware.

<https://www.olimex.com/Products/AVR/Programmers/AVR-ISP500/>

Once your software and drivers are installed, you can use Atmel Studio.

1. Open Atmel Studio 6.0 and the programmer dialog box “Tools -> Device Programming”
2. Select Tool/Device/Interface and then click “Apply”
3. Select “Production file” on the left bar
4. Select the elf file and the “Flash” checkbox and then click “Program” button

Screenshots in Figures 8-9 illustrate the process.

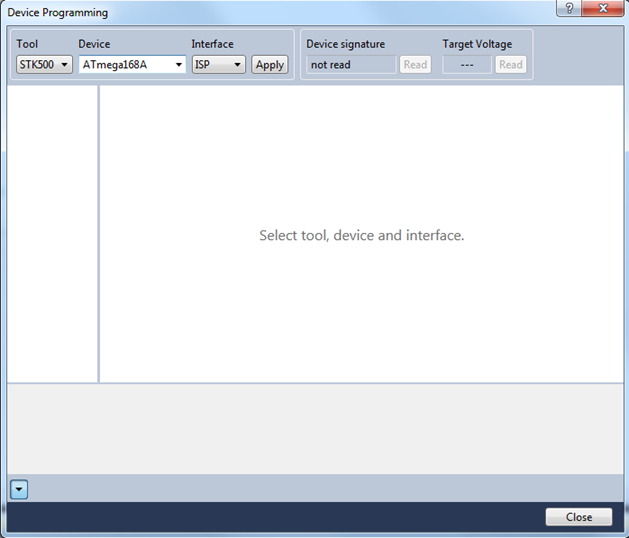


Figure 8 - Atmel Studio Device Programming dialog.

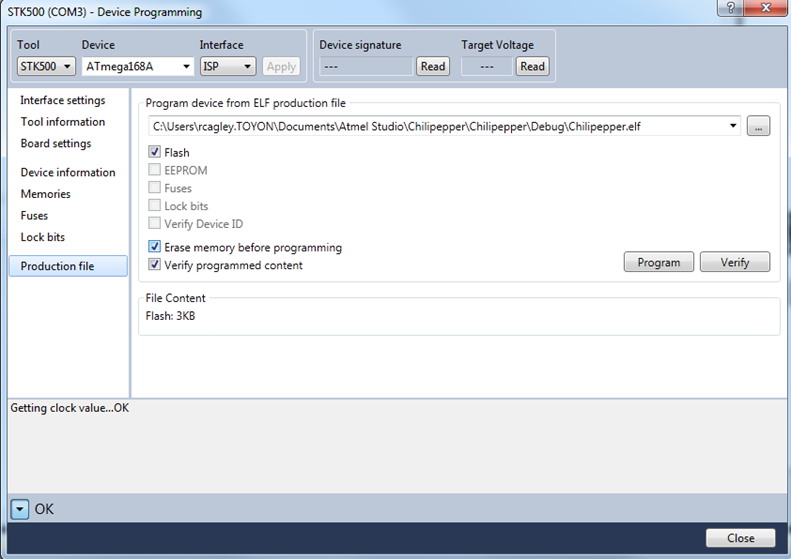


Figure 9 – Production File programming box.

# Precautions

* Always use proper ESD handling precautions.
* When no antenna connected please attach 50 ohm passive load.

# Reference Designs

There is Quadrature Phase Shift Keying (QPSK) example design and set of laboratory exercises. Please navigate to github to peruse the work.

<https://github.com/Toyon/Chilipepper>

# References

1. R. E. Cagley, S. A. McNally, B. T. Weals, R. A. Iltis, S. Mirzaei, and R. Kastner, "Implementation of the Alamouti OSTBC to a Distributed Set of Single-Antenna Wireless Nodes", in Proc. IEEE Wireless Communications and Networking Conference, Hong Kong, Mar. 2007.
2. R. E. Cagley and Nicholas Hogasten, “Bridging the Gap Between Advanced Image Processing and Hardware Design,” Synopsys Users Group Conference, June 2010.
3. R. E. Cagley, “Automating FPGA/ASIC Design Workflow with MATLAB”, The MathWorks Signal Processing Virtual Conference, May 2011.
4. FAQ

[Q1] In my signal from the ADC I see a DC offset. How do I remove it?

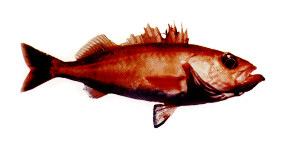
[A1] The best way is to implement a carrier offset and then apply a high pass or bandpass filter. This can easily be done with a cascade-integrator-comb (CIC) filter. Alternatively if you want to do direct-conversion you can simple track and remove the DC bias, e.g., dc = (1-alpha)\*dc + alpha\*adc\_signal.

[Q2] What does the flashing LED on Chilipepper mean?

[A2] This indicates that either your dip switches are not in a valid mode or that Chilipepper has not been correctly configured. Please check the dip switches and ensure that proper control signaling is followed according to Section 3.

1. Board Revisions
2. What is a Chilipepper?

We name all our boards after fish because several of us fish. Chilipepper is the common name for *Sebastes goodei* from the family *Scorpaenidae*, which is a deep water rockfish found off the California coast. It is normally found over rocky bottoms and feeds on small crustaceans, squid, and fishes. It can live up to 16 years and opposite from most fish, it gives birth to live young. The current record Chilipepper is 22 inches long and 5.25lbs. Luckily your Chilipepper is not that big.



1. Example Constraints





1. For supported bandwidth cutoff filter frequencies, see page 5 of the following guide <http://www.limemicro.com/download/LMS6002Dr2-Programming_and_Calibration_Guide-1.1r1.pdf> [↑](#footnote-ref-1)