Hes·so VALAIS WALLIS
School of Engineering

浙江大学 ZHEJIANG UNIVERSITY

Degree Programme
Systems Engineering

Major Power & Control

Bachelor's thesis
Diploma 2018

Balle Marcel

Miniaturized Doppler radar sensor for non-contact vital sign detection

Professor
Andersson Alexandra

Expert
Lixin Ran

Submission date of the report
09.10.2018

HES-SO Valais

SYND	ETE	TEVI
X	X	X

Données du travail de diplôme
Daten der Diplomarbeit

FO 1.2.02.07.CB
haj/11/01/2016

Filière / Studiengang **SYND**	Année académique / *Studienjahr* **2017/18**	No TD / *Nr. DA* **pc/2018/46**
Mandant / *Auftraggeber* ☐ HES—SO Valais ☐ Industrie ☒ Etablissement partenaire *Partnerinstitution*	Etudiant / *Student* **Marcel Balle** Professeur / *Dozent* **Alexandra Andersson**	Lieu d'exécution / *Ausführungsort* ☐ HES—SO Valais ☐ Industrie ☒ Etablissement partenaire *Partnerinstitution*
Travail confidentiel / *vertrauliche Arbeit* ☐ oui / ja [1] ☒ non / nein	Expert / *Experte* *(données complètes)* **Lixin Ran** Zhejiang University, Hangzhou, P.R. China	

Titre / *Titel*

Miniaturized Doppler radar sensor for non-contact vital signs detection

Description / *Beschreibung*

The target of this project is to design and implement a miniaturized Doppler radar sensor for non-contact vital signs detection. In order to obtain a high sensitivity, the system is suggested to be designed based on highly integrated 60-GHz ISM-band transmitter and receiver. The sensor will consist of a pair of antennas, radio frequency (RF), digital intermediate frequency (IF) and baseband parts. A simple software will be developed to process the Doppler signals in digital domain based on existing algorithms. An experimental wireless detection of human heartbeats can be demonstrated using the implemented radar sensor.

Signature ou visa / *Unterschrift oder Visum* Responsable de l'orientation / filière *Leiter der Vertiefungsrichtung / Studiengang:* ..... [1] Etudiant / *Student* : Balle Marc	Délais / *Termine* Attribution du thème / *Ausgabe des Auftrags:* **16.05.2018** Remise du rapport / *Abgabe des Schlussberichts:* **À définir** Défense orale / *Mündliche Verfechtung:* **Semaine 42**

[1] *Par sa signature, l'étudiant-e s'engage à respecter strictement la directive DI.1.2.02.07 liée au travail de diplôme.*
Durch seine Unterschrift verpflichtet sich der/die Student/in, sich an die Richtlinie DI.1.2.02.07 der Diplomarbeit zu halten.

Rapport reçu le / *Schlussbericht erhalten am* ..... Visa du secrétariat / *Visum des Sekretariats* .....

Bachelors thesis Diploma 2018

Miniaturized Doppler radar sensor for non-contact vital signs detection

Marcel Balle

October 9, 2018

Student:	Marcel Balle	.....
Lab Director:	Prof. Lixin Ran	.....
Thesis Director:	Dr. Alexandra Andersson	.....
Date:	October 9, 2018	

I Abstract

Non-contact vital sign detection based on Doppler radar systems presents several advantages in bio-medical and healthcare applications. Many research efforts have been proposed in this field to obtain accuracy and effectiveness for reliable wireless bio-signal detection. However front-end architectures, demodulation algorithms and signal processing methods can be further improved to retrieve the heartbeat and respiration spectra. In this work, based on existing millimeter wave systems, a continuous wave Doppler radar sensor operating at 60GHz was designed and implemented utilizing a low-IF receiver architecture to accurately detect heartbeat and respiration rate of a human target. In order to achieve robustness in addition to effectiveness, the sensor is integrated on a miniaturized single-board and the components power consumption is low. Potential other applications for this radar system will also be discussed.

Index Terms - Doppler radar, vital signs detection, ISM-Band, low-IF

II Acknowledgement

I would like to express my most heartfelt gratitude to my supervising professor in Switzerland Dr. Alexandra ANDERSSON and my responsible professor of the project Prof. LIXIN Ran at Zhejiang University in Hangzhou.

I would also like to thank ZHANG Yi for assisting me upon my arrival in China, and all the students at the Advanced Radiofrequency Engineering (ARE) Laboratory that offered their support during my work.

Finally, I would like to thank Martial GEISER for giving me the opportunity to accomplish my Bachelor project abroad and helping me getting in touch with Zhejiang University.

Contents

III List of tables

IV List of figures

V List of Abbreviations

ADC	Analog to digital converter
ARE	Advanced Radiofrequency Engineering
CW	Continuous wave
DC	Direct current
DRS	Doppler radar system
DSP	Data signal processing
GPIO	General purpose input/output
HAL	Hardware abstraction layer
HSE	High speed external (clock)
I	In-phase
I/O	Input/output
IC	Integrated circuit
IF	Intermediate frequency
IR	Image reject
ISM	Industrial, Scientific and Medical
LDO	Low dropout regulator
LO	Local oscillator
LPF	Low-pass filter
LSB	Least significant bit
MCU	Microcontroller unit
MSB	Most significant bit
NF	Noise figure
PCB	Printed circuit board
PLL	Phased locked loop
OSC	Oscillator
Q	Quadrature
RF	Radio frequency
RSSI	Received signal strength indicator
SNR	Signal-to-noise ratio
SPI	Serial peripheral interface
SSB	Single-sideband
SWIM	Single wire interface module
TCXO	Temperature compensated crystal oscillator
UAV	Unmanned aerial vehicles
USART	Universal synchronous and asynchronous receiver-transmitter
USB	Universal serial bus
VCO	Voltage controlled oscillator
VGA	Variable gain amplifier
VSD	Vital sign detection
WLBGA	Wafer level ball grid array

1 Preamble

1.1 Introduction

The detection of human motions by wireless detectors has been developed through various methods including ultrasound, infrared and vibration. These simple detectors are commonly used in households and public places to automatically control lights, temperature, alarms, toilet flusher and automatic doors. Some wireless sensors are capable of providing information about position, direction, acceleration and velocity of the moving object, which can be used for measurement and analysis. Active non-contact sensors use microwave detection by emitting a continuous-wave signal which is reflected off surrounding objects and returns to the detector. If the signal hits a moving target, it's reflected frequency is altered. The phase shift in the returning radar microwave is measured and thus parameters of the targets motion can be extracted. This specialized detection process is also known as the Doppler Effect.

The use of Doppler radar systems (DRS) in healthcare and bio-medical applications has been widely studied due to its potential for non-contact bio-signal monitoring while eliminating the need of wired sensors attached to the subject. Non-contact detection of human vital signs, such as heartbeat and respiration rate, can be of significant improvement in applications like baby monitoring, rescuing earthquake survivors and tumor tracking. While several wireless sensors have already been implemented and shown successful results, the technology in this field can further be improved to obtain lighter, smaller and cost-effective modules with longer detection range and less power consumption in addition to achieving more accurate measurements.

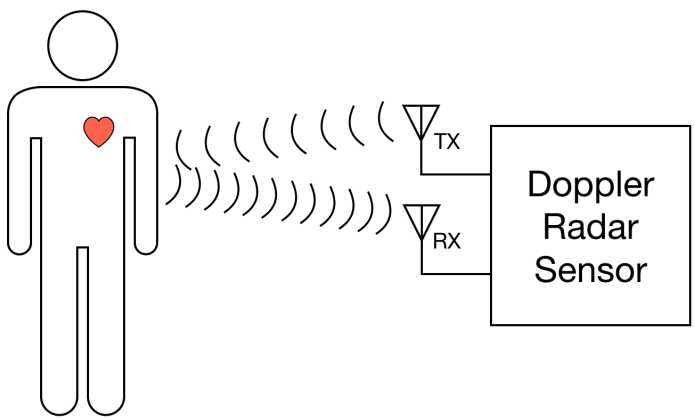


Figure 1: Representation of a DRS for vital sign detection

1.2 Goal

In this project, a miniaturized continuous-wave Doppler radar sensor has to be designed, implemented and tested for non-contact vital sign detection (VSD). In order to obtain high sensitivity, the system is suggested to be designed based on highly integrated 60-GHz ISM-band transmitter and receiver. The sensor will consist of a pair of antennas, radio frequency (RF), digital intermediate frequency (IF) and baseband parts. A simple software will be developed to process the Doppler signals in digital domain based on existing algorithms. An experimental wireless detection of human heartbeats can be demonstrated using the implemented radar sensor.

2 Description

2.1 Theory

2.1.1 Doppler effect

The Doppler effect was proposed by Christian DOPPLER in 1842 and confirmed later when tested with sound and electromagnetic waves. This effect is caused when a wave emitting source is moving relatively to the observer. If the frequency of the source and the propagation speed of the waves are constant, the observer will intercept the wave crests at different intervals due to the change of distance of the source. The gap between each wave varies, altering the wavelength. This leads to a frequency shift at the static receiving point. If the wave source is moving toward the observer, the distance between the two points becomes closer and the received frequency is increasing. In the case where the source is moving away from the observer, the frequency is decreasing. This same phenomenon occurs when the source is static and the observer is moving. The Doppler effect can result from any types of wave, i.e., sound waves, water wave, light wave, etc.

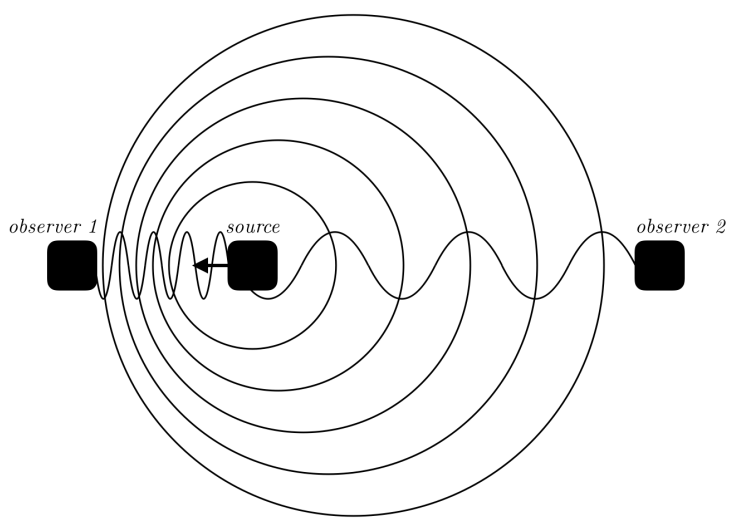


Figure 2: Simple representation of the Doppler effect

2.1.2 Doppler radar

The Doppler radar sensor is the source and at the same time the observer of the electromagnetic waves. The transmitter (T) generates a continuous electromagnetic millimeterwave with a sinusoidal form, where amplitude and frequency are fixed. This signal is transmitted in direction of the moving target. The frequency shift is then observed by the receiver (R) when the signal is scattered off the moving object and returns to the sensor. With this information, it is possible to measure the velocity of the moving target.

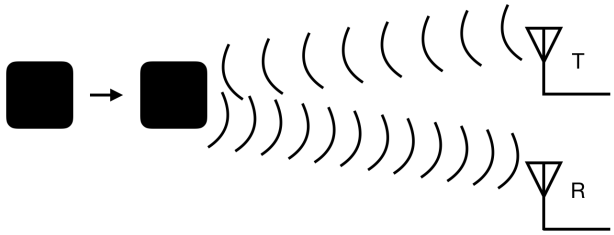


Figure 3: Frequency shift caused by moving object

Radar type In order to obtain the Doppler effect from a target, the antenna has to radiate a type of signal. Most Doppler radar sensors planned for vital sign detection use unmodulated continuous wave (CW) technology because it is small, light and simple. The transmitted constant signal is fixed in frequency and amplitude, making its generation uncomplicated and automated. CW radars are not able to provide range information as there is no way to measure the time delay between the send and the returned wave. As a result, stationary targets don't cause any changes to the electromagnetic signal and the extracted Doppler shift only provides data about the moving objects. The transmitted power is low thus no high voltage modulators are required for simple CW radars and no interference occurs with other wireless systems.

Radar frequency Continuous wave radars transmit one stable radio frequency which can be chosen in the Industrial, Scientific and Medical (ISM) bands. In most countries, these bands of frequencies can be used for any purpose without the need of a license. Systems that adapt the standards and regulations can be used worldwide with little or no adjustment to their implementation. The spectrum of electromagnetic waves is large and can be divided into sub-ranges depending on the different physical qualities. The frequency of a radar system should be determined by its predefined operating band using the range sections as seen in Fig. 4. The accuracy of the DRS increases with the transmitted frequency.

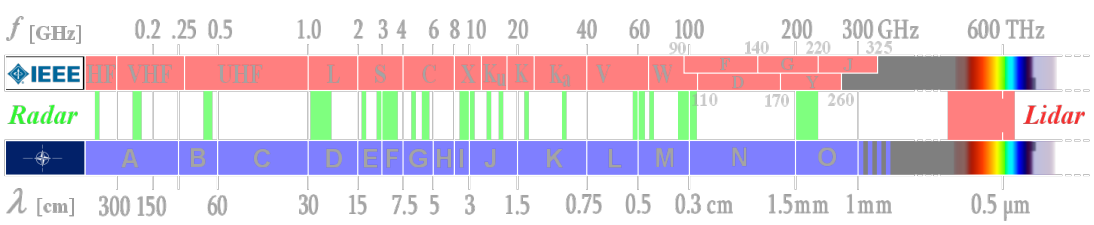


Figure 4: Waves and frequency ranges used by radar [1]

2.1.3 Frequency variation

A Radar is emitting a continuous wave with frequency f_t at a moving object with velocity v and angle θ placed at distance R from the sensor as seen in Fig. 5.

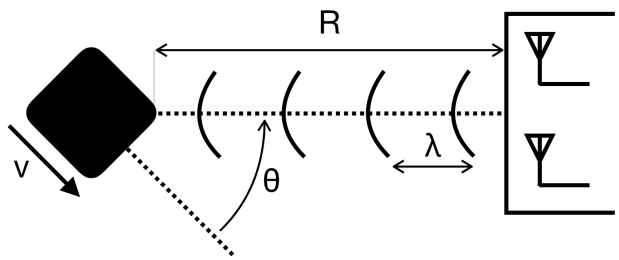


Figure 5: Angle of moving target influence on frequency shift

On the path traveled by the signal before being received, the number of wavelengths λ is given by n

$$n = \frac{2R}{\lambda} \tag{1}$$

The transmitted signal from the sensor has a sine form, therefore one wavelength has an angular excursion of 2π. The angular excursion ϕ of the entire path by the wave is

$$\phi = \frac{2R}{\lambda} \cdot 2\pi = \frac{4\pi R}{\lambda} \tag{2}$$

As a result of the object moving toward or away from the sensor, the distance R is constantly changing, meaning that ϕ relative to time is equal to the Doppler angular frequency ω_d as seen in [2].

$$\omega_d = 2\pi f_d = \frac{d\phi}{dt} = \frac{4\pi}{\lambda} \cdot \frac{dR}{dt} = \frac{4\pi \cdot v \cos\theta}{\lambda} \tag{3}$$

Knowing that

$$\lambda = \frac{c}{f_t} \tag{4}$$

with c the speed of light, the Doppler frequency (or Doppler shift) f_d which is defined as the difference between the received frequency f_r and the transmitted frequency f_t, can be described by the following equation.

$$f_d = f_r - f_t = f_t \frac{2v\cos\theta}{c} \tag{5}$$

Note that the Doppler frequency is positive when the moving object is approaching and negative when it is moving away from the radar. If the object is moving perpendicular to the radars line-of-sight (i.e., $\theta = \frac{\pi}{2}$), the Doppler frequency equals zero. Contrarily the Doppler frequency is a maximum when $\theta = 0$.

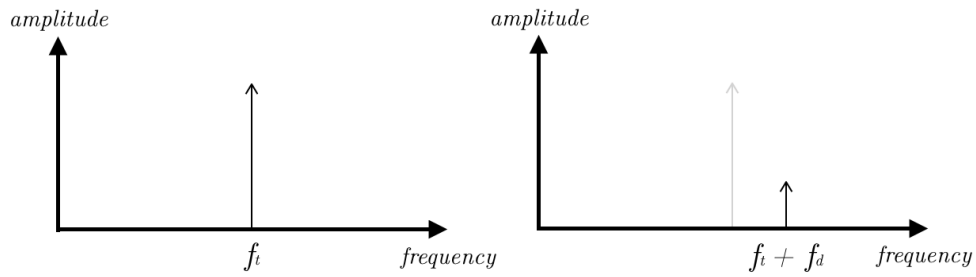


Figure 6: Frequency spectrum of the transmitting (left) and the receiving (right) antenna

2.1.4 Doppler shift

An unmodulated continuous waveform signal T_{RF} is transmitted towards the human target, phase modulated by the chest motion and then the reflected returning signal R_{RF} is captured by the receiving antenna. These two signals can be represented as

$$T_{RF}(t) = A(t)cos\left[2\pi f_c t + \phi(t)\right]$$
$$R_{RF}(t) = A(t)cos\left[2\pi f_c t - \frac{4\pi d_0}{\lambda} - \frac{4\pi x(t)}{\lambda} + \phi(t - \frac{2d_0}{c})\right]$$

where

- $A(t)$ is the signal amplitude whose variation is neglected,
- f_c is the the carrier frequency,
- $\phi(t)$ and $\phi(t - 2d_0/c)$ is the phase noise,
- $4\pi d_0/\lambda$ is the phase delay due to the distance between the antennas and the reflection surface d_0,
- $4\pi x(t)/\lambda$ is the phase modulation by the periodic chest motion $x(t)$ or as mentioned above it is the Doppler angular frequency ω_d,

In applications like healthcare monitoring, i.e. respiration rate and heartbeat measurements, the Doppler shift produced by the chest and heart movements of the target is relatively low compared to the carrier frequency. In this case, it is difficult to extract precise and correct vital frequencies by using the Doppler shift. So, in order to obtain accurate measurements, the phase shift θ_0 from the downconverted $R_{RF}(t)$ equation is used to determine the desired frequencies. Section 3.2 describes the I/Q signals including the phase shift θ_0 after downconversion and demodulation.

2.1.5 Doppler radar system

The Doppler radar sensor that has to be designed transmits a continuous wave signal with stable frequency f_t in the direction of the target through a transmitting antenna. After being reflected off various objects in the environment, the signal returns near the DRS and is measured by the receiving antenna. If the microwave signal is scattered off a moving target, the returning signal frequency f_r will be deviated from the RF by the Doppler frequency f_d seen in the previous section.

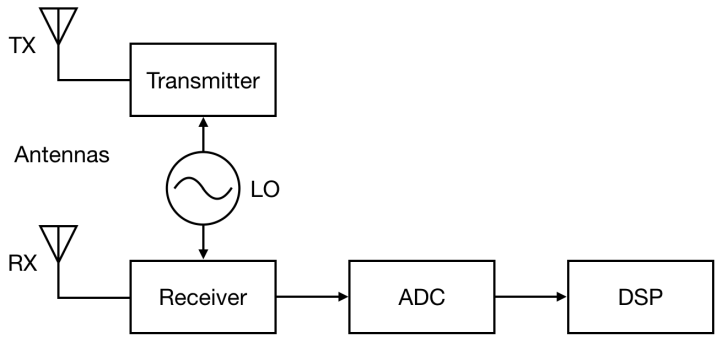


Figure 7: DRS simplified block diagram

A Doppler radar system incorporates a transmitter, which is the source of the continuous wave radio frequency, and a receiver picking up the reflecting signal. They can be combined into one part named Transceiver. Both devices need a common external local oscillator (LO) as an input to create and extract the desired frequencies. Two antennas, isolated from each other to keep the transmitter leakage low, convert electrical signals into electromagnetic waves for the transmitter and the other way around for the receiver. The received signal is down-converted to baseband or an intermediate frequency carrying the relative velocity information. Then this transformed signal is digitized by analog to digital converters (ADC) to proceed with data signal processing (DSP). Fig. 7 shows a simplified block diagram of a Doppler radar system.

2.2 Design

The entire DRS is consolidated in a single printed circuit board (PCB) in order to reduce the size of the sensor. The board is connected to a computer via universal serial bus (USB) for data transferring and power supply. Alternatively, a supplementary connector mounted on the PCB is designed to link a Bluetooth module which communicates the required data with a computer. The transmitting and receiving 60-GHz signals antennas are integrated on the board as series-fed microstrip patches. Only data transferring and power supply for all the parts are connections from outside the PCB. A Microcontroller unit (MCU) is used to convert the received downconverted signal in digital domain and then extract the measured data. Management and information control of the programmable chips on the PCB is also accomplished by the MCU.

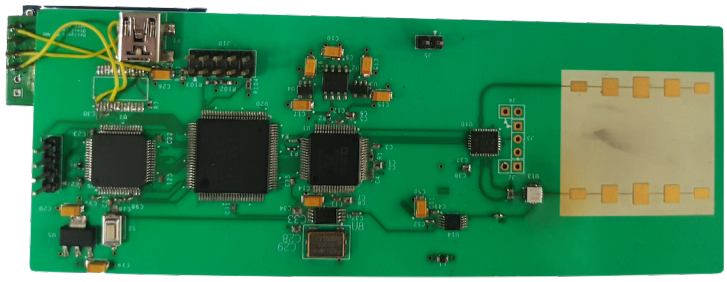


Figure 8: Example of a 24GHz DRS board for VSD

2.3 Objectives

The fundamental goal of this project is to understand the main functioning of a microwave radio frequency radar. Furthermore such a sensor has to be designed and implemented for specific application in vital sign detection. Described in this section are the individual objectives for every part of this work.

Electronic In this part of the project the radar structure has to be designed and implemented. The power consumption of the sensor is considered as low as possible so that the system can be supplied by a small battery and be able to function during an extended period of time without needing supervision.

Mechanical The PCB board holding all the Integrated circuit (IC) chips, active and passive components as well as the antennas, should be as minimal in size as possible. The purpose of this requirement is to assure a space efficient and discrete measurement device.

Data processing In order to be accurate and thus reliable, the acquired data from the DRS is processed by efficient algorithms. Objectively, the results of the measurements have to be precise enough so that non-contact vital sign detection Doppler radars can be considered as a replacement for typically used wired sensors.

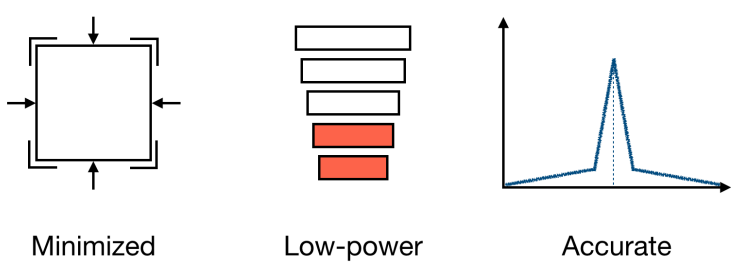


Figure 9: Simple representation of the main objectives

2.4 Tasks

Based on the objectives described above, a list of tasks is decided for this project. Note that not all the parts have to be completed in the order of the list in Table 1.

ID	Task
1	Determine architecture
2	Choose components
3	Draw schematics
4	Create layout
5	Produce PCB
6	Write software
7	Implement algorithms
8	Run tests

Table 1: List of tasks

2.5 Domain of use

By accurately measuring respiration rate and heartbeat of a human subject, VSD radars can be of great advantages in medical applications to constantly monitor patients, such as infants and elderly in their hospital rooms as well as in their homes. Non-contact detection technology eliminates the need of wired sensors attached to the patient, therefor no disturbance or discomfort is caused to the subject while taking measurements. Many studies have been lead in this matter such as tumor tracking in cancer radiotherapy [3] and vital sign monitoring inside an office or an automobile [4], [5] using Doppler radar technology.

Nowadays, some smartphones are capable of reproducing such a Doppler effect. This would mean that human vital sign detection and measurement is possible from a commonly used device, reducing device transit time and cost. In [6] a non-contact cardiopulmonary detection method in natural disaster rescue relief, based on the use of frequencies and antennas that are similar to current smartphone-like devices is explored. This research matter has a lot of potential as smartphone technologies are increasing drastically and these devices are commonly used worldwide.

3 Architectures

3.1 Principal components

Oscillators This component generates an electronic wave signal at a desired frequency. It is used to create the radio frequency and the intermediate frequency in a DRS. Oscillators can be voltage controlled (VCO) or crystals, where the signal originates from a quartz crystal using the Piezoelectric effect.

Mixers The frequency mixer is used to multiply two signals with each other. If these signals have sinus forms, the output is sum and difference of the two input frequencies. The output signal of unbalanced mixers consists of the mixed product signal as well as both input frequencies. Balanced mixers only make the output contain one of the entering signals. Real mixers produce sum and difference of the two input frequencies while complex mixers only add them together.

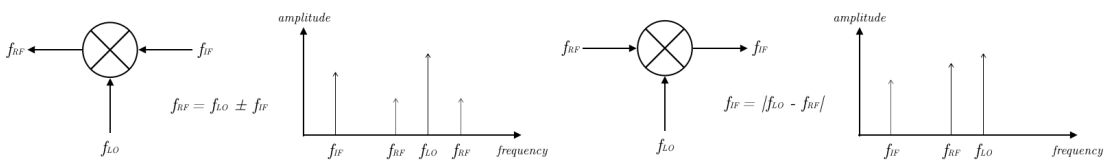


Figure 10: Mixer upconversion and downconversion frequency spectrum

Amplifiers Low-noise amplifiers (LNAs) increases the power of the selected signal while adding as little noise and distortion as possible. These components are characterized by their gain and their Noise figure (NF).

Filters Used to emphasize signals in a particular frequency range while rejecting or suppressing those in the undesired frequency range. Filters can attenuate higher frequencies (low-pass filter) or lower frequencies (high-pass filter) than a desired frequency. Band pass and band stop filters suppress frequencies outside and inside a specified band respectively.

Analog to digital converters ADCs transform analog electrical signals into digital impulses in order to process the collected data by a microcontroller. This component is characterized by its sampling rate, which should be faster than the input frequency and its signal-to-noise ratio (SNR).

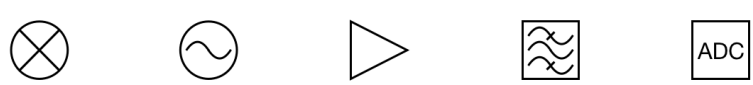


Figure 11: Symbols of Oscillator, Mixer, Amplifier, Filter and ADC

3.2 Receiver Architecture

The transmitter from a wireless RF system has simple tasks to accomplish, thus its circuitry remains basic. The receiver architecture on the other hand adapts various designs suitable for specific applications. The most commonly used architectures are briefly presented below.

3.2.1 Heterodyne

The most commonly used receiver in wireless radar systems is the super-heterodyne architecture invented by Armstrong in 1917, Fig. 12. The transmitted RF signal is the up-mixing product of a defined intermediate frequency with a local oscillator. Both signals serve to downconvert the returned signal in the receiver.

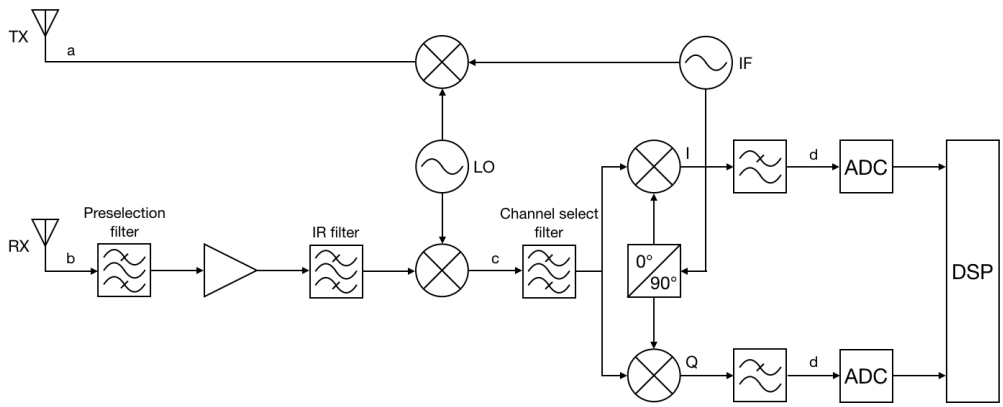


Figure 12: Block diagram of heterodyne architecture

The first preselection filter removes out-of-band signal energy and partially rejects image band signals from the received RF signal before being amplified by the low-noise amplifier to suppress the contribution of noise from the following stages. A band-pass image reject filter (IR) attenuates the image frequency effects coming from LNA. The filtered signal is then downconverted from RF frequency to IF frequency in the first mixer using the output of the LO. After this stage, another IR filter selects the desired channel by rejecting undesired IF bands and the signal is demodulated and downconverted to direct current (DC) by mixing In-phase (I) and Quadrature (Q) components with the IF frequency source used in the transmitter. Low-pass filters (LPFs) serve as channel reject filters as well as anti-aliasing functionalities. The I/Q signals are finally converted to the digital domain to retrieve the desired information through data processing.

For this front-end architecture the carrier frequency is equal to $f_c = f_{IF} + f_{LO}$ and the phase noise $\phi(t) = \phi_{IF}(t) + \phi_{LO}(t)$. Then the transmitting and the receiving signals can be presented as the equations seen in section 2.1.4. After mixing the local oscillator frequency with the filtered received radio frequency, the signal at point c including the phase shift θ_0 at the reflection surface can be analyzed as

$$R_{IF}(t) = f_{LO}(t) - R_{RF}(t) = A(t)cos\left[\frac{4\pi d_0}{\lambda} + \theta_0 + \frac{4\pi x(t)}{\lambda} - 2\pi f_{IF}t - \phi_{IF}(t - \frac{2d_0}{c}) + \phi_{LO}\frac{2d_0}{c}\right]$$

According to range correlation $\phi(t - 2d_0/c) \approx \phi(t)$, then the demodulated and to baseband downconverted I/Q components on point d can be written as

$$I(t) = A(t)cos\left[\frac{4\pi d_0}{\lambda} + \theta_0 + \frac{4\pi x(t)}{\lambda}\right]$$
$$Q(t) = A(t)sin\left[\frac{4\pi d_0}{\lambda} + \theta_0 + \frac{4\pi x(t)}{\lambda}\right]$$

Fig. 13 illustrates the frequency spectra of the transmitted RF signal produced by mixing IF and LO frequencies as well as of the received signal with a positive Doppler shift. The frequency spectra of the signals at IF and baseband after being mixed with the initial LO and the IF source respectively are also displayed.

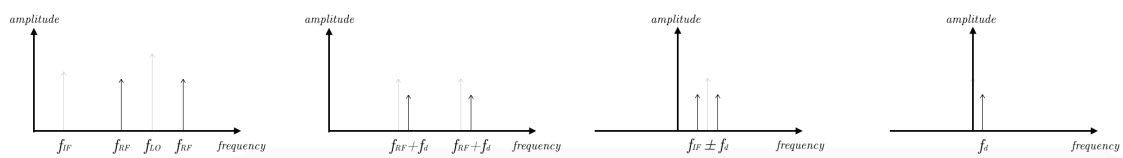


Figure 13: Frequency spectra of a) transmitted RF b) received RF c) signal downconverted to IF d) signal downconverted to DC

This architecture is adopted in most microwave systems as the IF frequency is fixed and reduced from high frequencies, optimizing the components power consumption and increasing their availability in low-cost. Additionally the heterodyne receiver can ensure good level of sensitivity, frequency stability and selectivity. Unlike homodyne architectures, DC-offset problems are absent in this structure. However this design presents some substantial disadvantages, like difficulties to eliminate undesired image frequency with image rejection filters and the need of high performance oscillators to reduce phase noise.

3.2.2 Zero-IF

Also known as homodyne or direct conversion receiver, the zero-IF architecture, shown in Fig. 14, is a simplified version of the heterodyne structure. It consists of one microwave signal source which generates the transmitted RF frequency and downconverts the received signal directly to baseband.

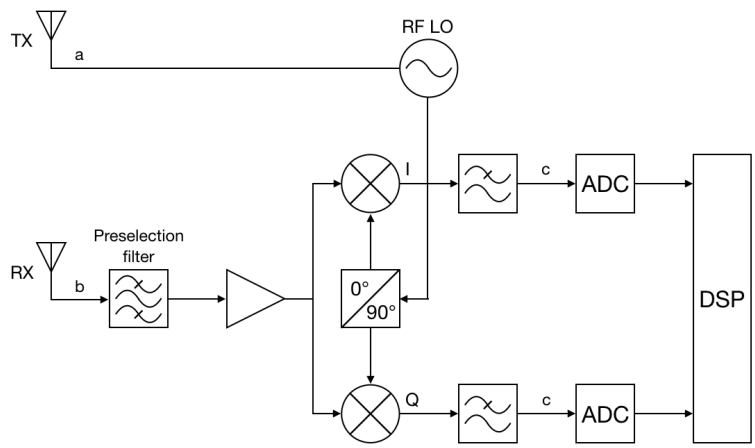


Figure 14: Block diagram of a zero-IF architecture

The received signal is selected at RF by a band-pass filter and then amplified by an LNA, as in the previous architecture. Next the output signal is directly downconverted from the carrier frequency to DC by two mixers with a delay of 90° between them to achieve I/Q demodulation. Quadrature downconversion is required for frequency and phase modulated signals to avoid loss of information, because in general the two side-bands of the RF spectrum are different. Low-pass filtering is applied in the baseband to suppress nearby interferes and select the desired channel. The baseband signal is finally converted to the digital domain and proceeds to DSP.

Similar to the the heterodyne architecture, the baseband in-phase and quadrature signals for the homodyne receiver are

$$I(t) = A(t)cos\left[\frac{4\pi d_0}{\lambda} + \theta_0 + \frac{4\pi x(t)}{\lambda}\right]$$
$$Q(t) = A(t)sin\left[\frac{4\pi d_0}{\lambda} + \theta_0 + \frac{4\pi x(t)}{\lambda}\right]$$

Fig. 15 shows the frequency spectra of the transmitted microwave signal, of the received signal with a positive Doppler shift and of the signal at baseband after being mixed with the initial LO.

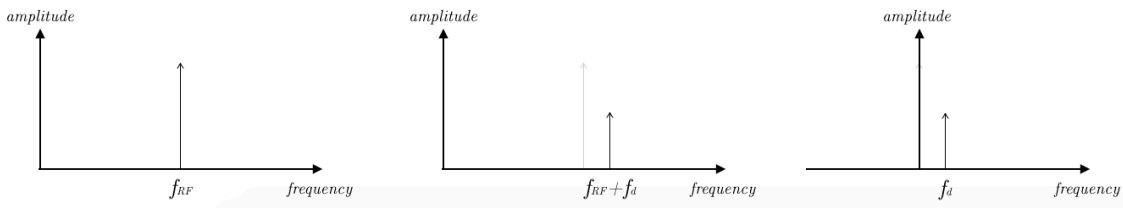


Figure 15: Frequency spectra of a) transmitted RF b) received RF c) signal downconverted to DC

Direct conversion provides several advantages compared to the other architectures. Its simple implementation demands less hardware, especially image reject filters (IR) are not required because no image problem are present. The main advantage of a homodyne system is range correlation resulting in cancellation of the oscillator phase noise. Despite its simplicity, this architecture includes problems as DC offset and LO leakage. Flicker noise created from direct downconversion to DC and baseband amplification can be of great concern when the produced Doppler shift is inside the 1/f noise range as seen in Fig. 16, making it impossible to filter unwanted frequencies and extract desired data.

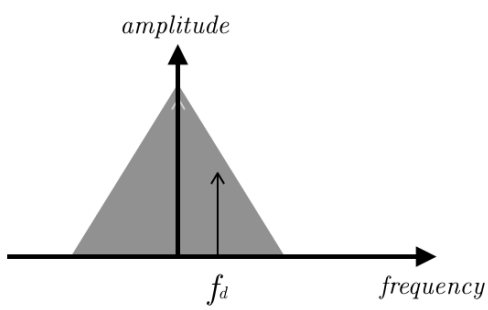


Figure 16: Frequency spectrum of downconverted signal with flicker noise

3.2.3 Low-IF

Another common architecture is the low-IF receiver, Fig. 17, which is a mixture between the heterodyne and the homodyne designs. Intermediate frequency and radio frequency are mixed together to create the transmitted signal. Only the local oscillator is applied on the returned signal for quadrature mixing and downconversion to low-IF.

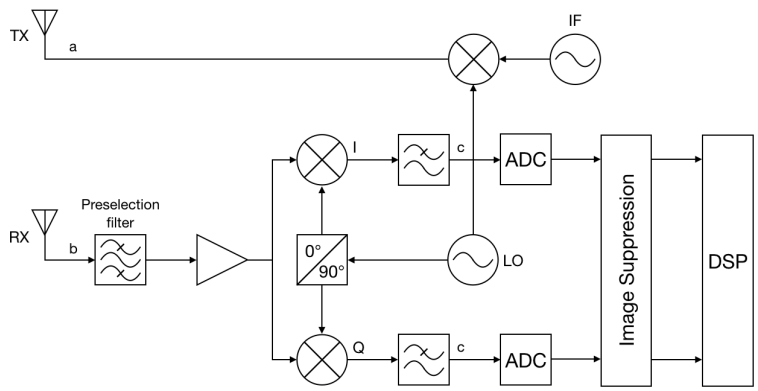


Figure 17: Block diagram of a low-IF architecture

Like zero-IF receiver, this architecture requires a channel-selection IR filter and a LNA at RF. After the first stage, all RF channels are complex mixed and downconverted to a fixed low IF instead of zero IF before passing through a low-pass filter. In order to eliminate the negative effects from frequency image an image suppression mixer is implemented either in analog or digital domain.

As for the heterodyne architecture, the carrier frequency is equal to $f_c = f_{IF} + f_{LO}$, thus the transmitting and the receiving signals can be presented as seen previously. After demodulation and downconversion to intermediate frequency, by using the range correlation simplification $\phi_{IF}(t - 2d_0/c) \approx \phi_{IF}(t)$ and $\phi_{LO}(2d_0/c)$ being neglected, the I/Q signals can be presented as

$$I(t) = A(t)cos\left[\frac{4\pi d_0}{\lambda} + \theta_0 + \frac{4\pi x(t)}{\lambda} + 2\pi f_{IF}t - \phi_{IF}(t)\right]$$
$$Q(t) = A(t)sin\left[\frac{4\pi d_0}{\lambda} + \theta_0 + \frac{4\pi x(t)}{\lambda} + 2\pi f_{IF}t - \phi_{IF}(t)\right]$$

Fig. 18 illustrates the frequency spectra of the transmitted RF signal produced by mixing IF and LO frequencies and of the received signal before as well as after being downconverted from RF to low-IF frequency in the receiver.

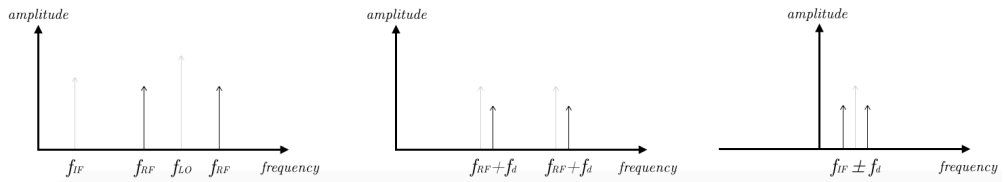


Figure 18: Frequency spectra of a) transmitted RF b) received RF c) signal downconverted to IF

Low-IF receivers architecture is similar to the homodyne and the heterodyne structure, however they present unique advantages and disadvantages. Due to the upconversion mixer, low-IF designs have to consider image suppression to reject unwanted symmetric frequencies. Since the IF frequency is not located at baseband, DC-offset is not an issue and flicker noise is absent in the measured frequency range, yet the ADC power consumption and cost is increased because a higher sampling rate is necessary to convert the non-zero IF frequency.

4 Hardware

4.1 Structure

Receiver architecture From the previous chapter the low-IF design is chosen for this miniaturized Doppler radar sensor for multiple reasons. As heartbeat and respiration have high sensitivity, meaning small frequencies and amplitudes, they generate a Doppler effect close to the emitted radio frequency. If the received signal is downconverted to baseband, as in direct conversion architectures, DC offset issues appear and the desired frequencies to analyze would be located in the flicker noise zone hence giving wrong measurements (as seen previously in Fig. 16). The low-IF receiver eliminates DC offset and avoids the region of highest flicker noise in the mixer output. Furthermore, by using the same source LO for the transmitted signal as for the downconversion of the received signal, the information about the periodic target motion can be demodulated. In this case the phase noise of the receiving signal and the LO is correlated proportional to the time delay between these two signals and thus the target range. This is called range correlation and is of great interest in vital sign detection since the phase noise in the low frequency range of heartbeat and respiration rate is very high. In a low-IF system the null point and I/Q channel imbalance problems are solved. The intermediate frequency is low enough to be digitized directly, the trade-offs are that this system requires high resolution, faster sampling ADCs and signal processing is heavier.

Carrier frequency As mentioned in the description of the project, the system is suggested to be designed based on highly integrated 60GHz ISM band transmitter and receiver. This carrier millimeterwave frequency allows a higher sensitivity to small movements by the target and a range detection of a couple of meters. Another advantage is the reduction of the antennas size allowing a miniaturized DRS design. However, vital sign detection consists of measuring two close-frequency signals with considerable amplitude difference, making it difficult to distinguish them. In order to improve the detection accuracy of heartbeat and respiration rate simultaneous analysis, complex phase demodulation algorithms have to be implemented in signal processing.

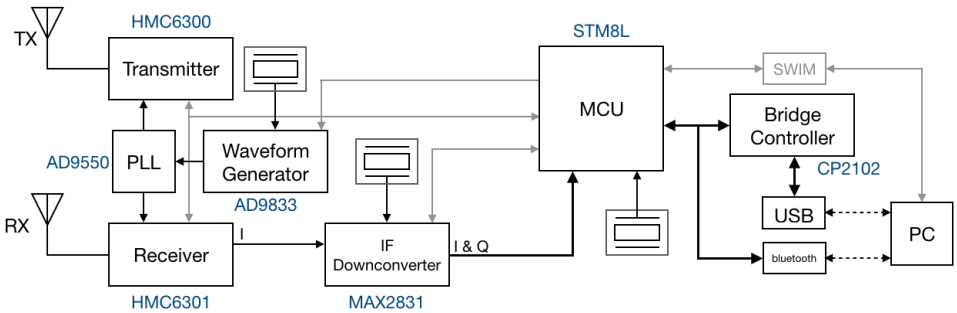


Figure 19: Miniaturized Doppler radar structure block diagram

The block diagram in Fig. 19 shows the structure chosen for the miniaturized CW Doppler radar. It consists of a transmitter and a receiver chip working with the same reference clock, established by a waveform generator and a clock translator, in order to achieve a coherent low-IF system. A RF transceiver, driven by a second external oscillator, downconverts the IF provided by the receiver to a lower intermediate frequency which can then be directly digitized by the internal ADCs of the microcontroller. The programmable MCU manages the chips using SPI and can communicate with external devices via a micro USB interface or via Bluetooth by plugging a Bluetooth module to the board. All the datasheets of the components used for this PCB figure in the appendices.

4.2 Main components

ID	Device	Manufacturer	Function
1	HMC6300	Analog Devices	Transmitter
2	HMC6301	Analog Devices	Receiver
3	MAX2831	Maxim Integrated	IF Downconverter
4	STM8L	ST Electronics	Microcontroller unit
5	AD9833	Analog Devices	Waveform Generator
6	AD9550	Analog Devices	Clock Translator
7	CP2102	Silicon Labs	USB-to-UART Bridge

Table 2: Main components used in the DRS

4.2.1 Transmitter & Receiver

The IC chips HMC6300 and HMC6301 are complete millimeterwave transmitter and receiver operating from 57 to 64 GHz. The chips are packaged in a wafer level ball grid array (WLBGA) compatible to standard surface mount technologies on PCBs.
A frequency synthesizer covers the RF in 250, 500 or 540 MHz steps with low phase noise that can support modulations of up to at least 64 QAM. The Output provides up to 16 dBm linear power, while an integrated power detector monitors the output power so it does not exceed the regulatory limits. The chips offer either analog control or digital control of the IF and RF gains. All functions are controlled via a Serial Peripheral Interface (SPI). Figures 20 & 21 illustrate the functional block diagrams of these two chips from their datasheets located as appendices.

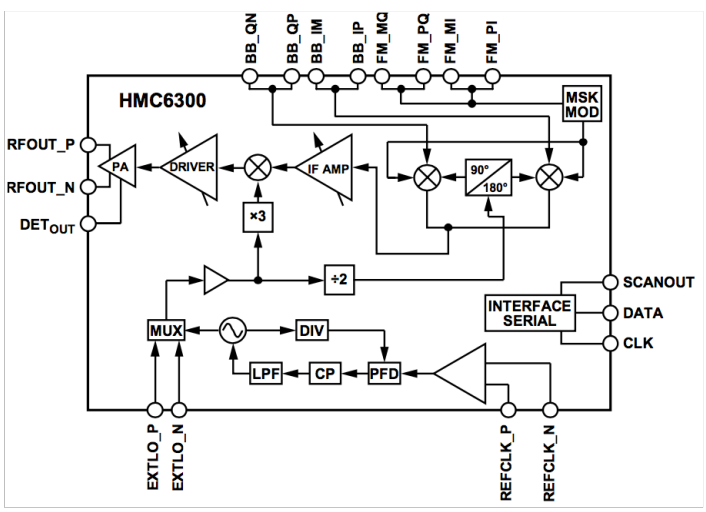


Figure 20: Functional block diagram of the transmitter chip HMC6300

Transmitter The integrated synthesizer consists of a voltage controlled oscillator (VCO), regulated by an on-chip phased locked loop (PLL) in order to obtain a low phase noise LO frequency. Half of that frequency is then complex-mixed with I/Q baseband signals, filtered and amplified. Finally the IF signal is upconverted to RF frequency between 57 and 64 GHz by using three times the LO frequency. Further filtering and amplification provide gain allowing differential output power with 22dB of variable gain. In single-ended configuration the output power includes 19dB of variable gain by disabling half of the power amplifier.

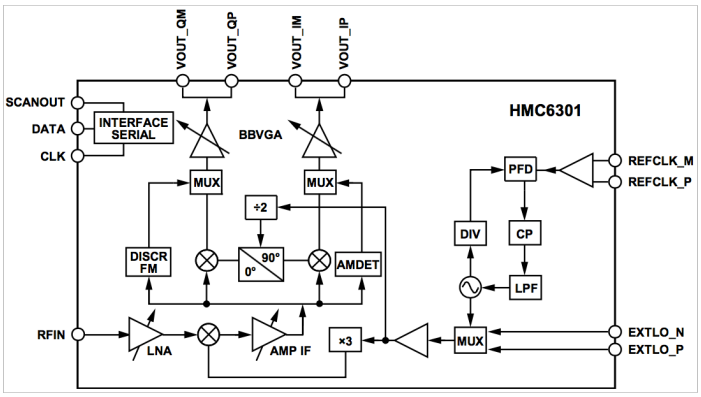


Figure 21: Functional block diagram of the receiver chip HMC6301

Receiver The received signal is provided with 20dB of variable gain using a single-ended LNA input. Instead of being upconverted, the RF is downconverted exactly the same way as in the transmitter to achieve the desired IF frequency at the differential quadrature outputs of the chip.

The step size of the synthesizer is proportional to the reference clock by 3.5. With a crystal of 71.42857MHz, the step size at RF equates to 250MHz. In this DRS a frequency of 70MHz is chosen for both chips in order to acquire a step size of 245MHz. With this configuration it is possible to obtain an IF of 2.45GHz at the output of the receiver to match the input frequency range of the subsystem downconverter. A DC signal is applied to the transmitters quadrature baseband inputs to exploit a single-sideband (SSB) transmission. Only the in-phase outputs of the receiver are used for further downconversion. Both ICs are regulated by the microcontroller writing to or reading from the register arrays using SPI.

4.2.2 Downconverter

The component used for the second IF downconversion is a single-chip, low-power, fully integrated RF Transceiver MAX2831 operating in the 2.4GHz to 2.5GHz band. As the MAX2831 is not exploited as a full transceiver, only the necessary pins and parts of the chip are operated to downconvert the 2.45 GHz output IF signal coming from the HMC6301.

The receiver part of this IC chip integrates an LNA and variable gain amplifier (VGA) with a 95dB digitally programmable gain control range, direct-conversion downconverters, I/Q baseband low-pass filters with programmable LPF corner frequencies, analog received signal strength indicator (RSSI), temperature sensor, Rx I/Q error-detection circuitry and integrated DC-offset correction circuitry. The block diagram with typical operating circuits (taken from the datasheet) is shown in Fig. 22.

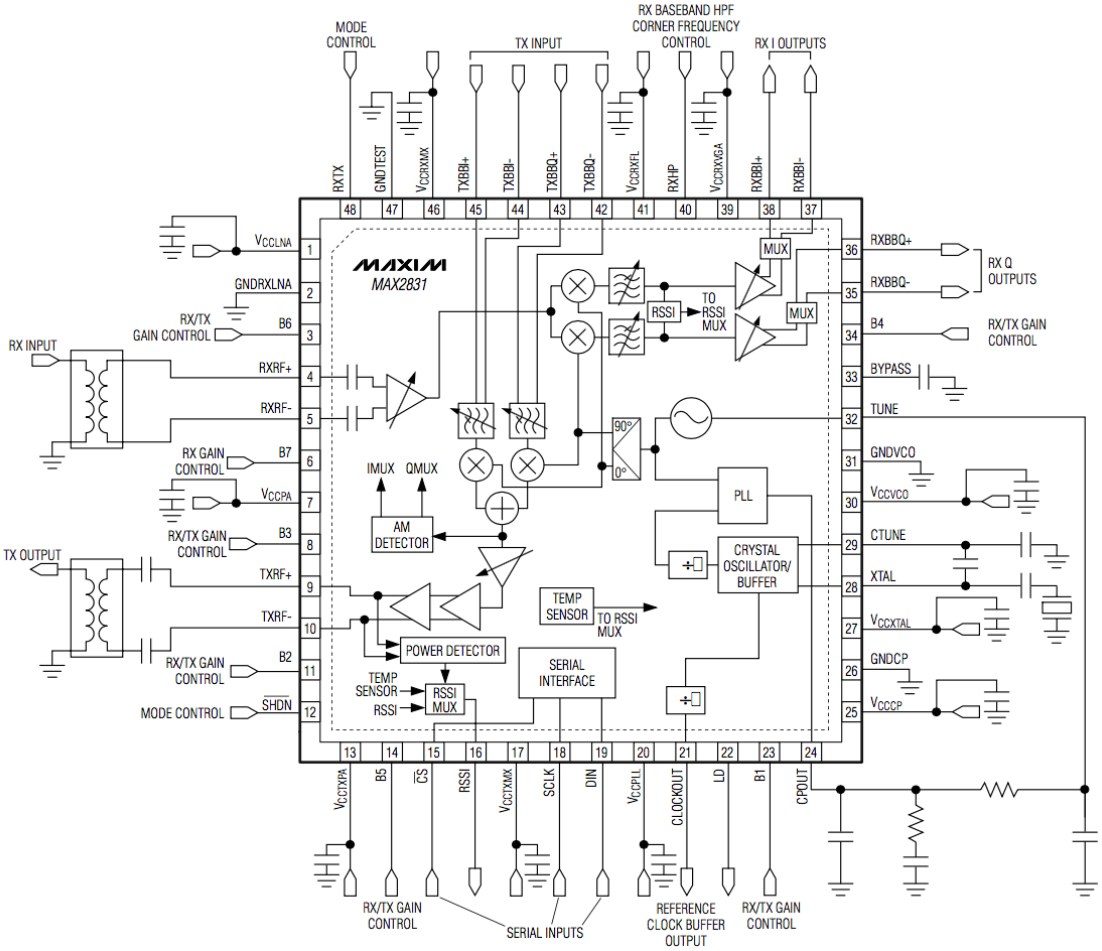


Figure 22: Block diagram of the transceiver chip MAX2831

A 3-wire SPI interface manages the programming of 16 registers containing 18 bits. These registers control LNA gain, baseband VGA, low-pass and high-pass filter corner frequencies, as well as adjustments of the common-mode output voltage and the information delivered by the RSSI output pin. For this radar system, the RSSI output is programmed to deliver an analog output voltage proportional to the received signal strength. In order benefit from the data of this signal, it is converted to digital domain in the MCU. The chip includes various modes of operation controlled by logic-input pins RXTX and SHDN. Shutdown mode allows the SPI registers to be loaded and Receive mode enables the receiver part of the IC. Other modes are described in the datasheet of the component (see appendices).

4.2.3 Microcontroller

In order to achieve the lowest power consumption, a microcontroller from the STM8L series is selected for this DRS. The STM8L151R8 includes 2 serial peripheral interfaces which provide half/full duplex synchronous serial communication with external devices and 3 universal synchronous and asynchronous receiver-transmitter (USART) interfaces used for data transmission between the board and an external PC via micro USB or Bluetooth. Furthermore, 54 individually configurable general purpose input/output (I/O) ports can be used for data transfer between the IC chips on the PCB. With an inbuilt 12-bit ADC containing up to 28 channels, it is possible to directly convert the downconverted I/Q signals for information extraction. Single wire interface module (SWIM) allows non-intrusive real-time in-circuit debugging and fast memory programming of the microcontroller. The chips pin connections to the other components on the board is shown in Fig. 23.

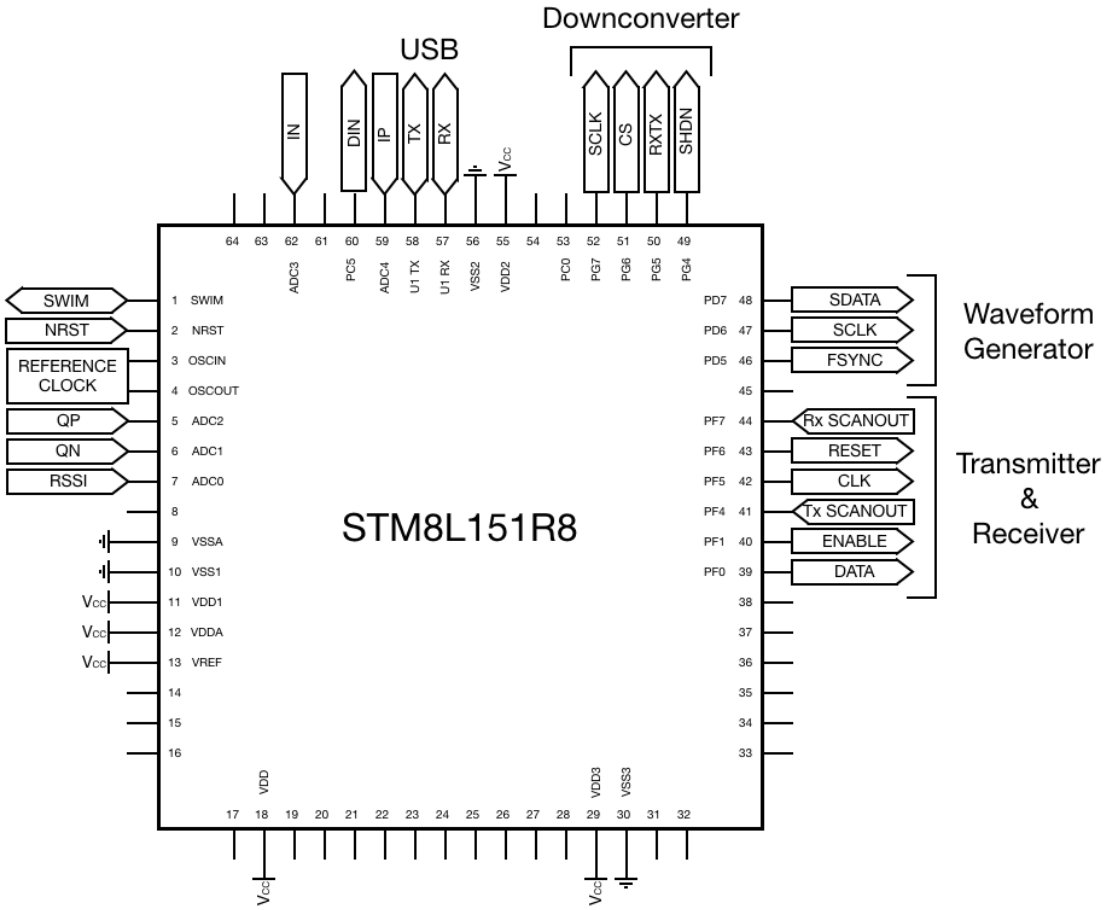


Figure 23: MCU pin connections

The device requires an external power supply chosen at 3.3 volts as well as ground connections. A 12MHz external crystal oscillator is used to drive the system clock through the OSC pins. SWIM and NRST ports serve for direct access to the debugging module and memory programming. For this design, the 5 first ADC channels manage the RSSI output and the differential I/Q outputs provided by the IF downconverter.

Transmitter and receiver share the same serial interface lines, as a write operation row includes the chips individual address. However for reading operations each component requires its own transmission line. General purpose I/O ports are used for the transmitter and receiver SPI lines as well as for programming the registers of the IF downconverter and the 3-wire serial interface of the reference clock generator chip. All the processed data is transferred via an USB to UART bridge or a Bluetooth module to a computer with the Tx and Rx ports on the STM8.

4.2.4 Oscillators

Transmitter and Receiver reference clock In order to minimize precision inaccuracies of the upconverted RF signal emitted by the HMC6300 and downconversion of the received signal by the HMC6301, both devices use the same reference clock. This will also contribute to coherence between the two chips. Furthermore, the reference clock of 70MHz is generated by a PLL based clock translator, a programmable waveform generator and a temperature compensated crystal oscillator (TCXO) as seen in Fig. 24.

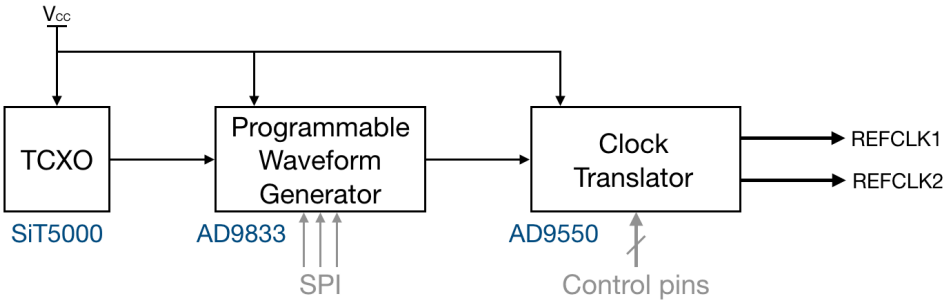


Figure 24: Components structure generating the reference clock

The TCXO delivers a stable frequency clock to the programmable waveform generator. The chip is capable of producing an 28-bit resolution output frequency up to 12.5MHz. With a crystal of 25MHz, resolution of 0.1Hz can be achieved. For this application, the AD9833 is programmed via the 3-wire SPI to output a frequency of 1.8436 MHz. This value is adopted for the clock translator in order to obtain two differential reference clocks at the exact frequency for the transmitter and the receiver chips. The output frequencies of the AD9550 are controlled by hardwired selection pins which set the prescalers, frequency dividers and multipliers as well as the PLL configurations. Fig. 25 demonstrates the input clock path and the corresponding selection pins logic states of the chip. For a Logic 0 the pin is connected to the ground while an open connection is decoded as logic 1. Further information on the frequency translation ratio from the reference input to the output of this chip can be found in the AD9550 datasheet located in the appendices.

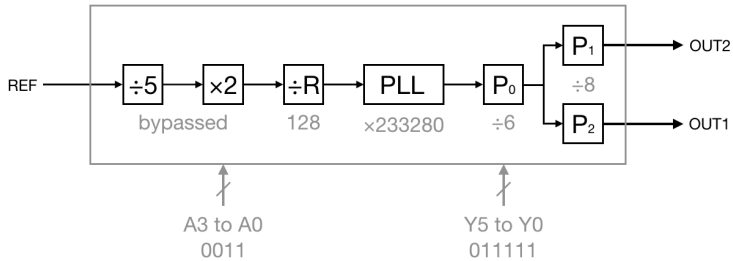


Figure 25: Input clock path with set pin logic and translation values

IF receiver crystal For the reference clock of the MAX2831 chip, another SiT5000 TCXO is used with a frequency of 40MHz corresponding to the RF transceivers reference frequency range. The oscillator is AC coupled to the XTAL analog input.

MCU oscillator To drive the master clock of the microcontroller, the high speed external clock signal (HSE) is generated by a surface mount crystal resonator delivering a frequency of 12MHz. Together with its load capacitors, the oscillator is placed as close as possible to the oscillator (OSC) pins in order to minimize output distortion and start-up stabilization time.

4.2.5 Micro USB and Bluetooth

Retrieved and converted information of the Doppler shift in the frequency response is transferred to an external computer for data processing either through micro USB connection or by a Bluetooth module directly plugged on the sensor PCB.

Micro USB The USB interface provides a power supply of 5V for every LDO, i.e., the power supply for the entire board. In order to convert the transmitter (Tx) and receiver (Rx) signals from the USART of the microcontroller to USB data for the micro USB connector, a single-chip USB-to-UART Bridge controller is implemented. The CP2102 is a simple solution for updating RS-232 designs to USB using a minimum of components and PCB space. The datasheet of this component can be found in the appendices. In the case where this type of data transmission is utilized, the board is connected to the computer by a simple USB cable.

Bluetooth module For this choice of data transmission, an external portable battery is connected to the micro USB on the board port to supply all the components. The Bluetooth module HC05 is plugged onto the connector on the PCB meant for that purpose. The connector is directly coupled to the USART transmitter and receiver pins of the microcontroller. For more information about this Bluetooth module, refer to its datasheet in the appendices.

4.2.6 Power management

Different voltages and maximum current consumption are required by most of the chip components and therefore low dropout regulators (LDOs) are implemented for all the necessary supply voltages on the PCB. These chips use the power supply delivered from the USB connector as input voltage. A list of the LDO devices used on the sensor board is represented in Table 3.

ID	Device	Output Voltage	Max current
1	ADM7172	Adjustable	2A
2	ADP7118	Adjustable	200mA
3	AMS1117	3.3V	1A

Table 3: LDOs used in the DRS

The only fixed output voltage is a standard value of 3.3 Volts. As for all the other supply values, they are adjusted from a 1.2V output with an external voltage divider. In the divider, the two resistors are determined according to the following equation from the ADP7118 and ADM7172 datasheets. Resistor R2 should have a value of less than $200k\Omega$ in order to minimize errors in the output voltage. In Table 4, every LDO is listed with its output voltage and the maximum current consumption of the supplied chips.

$$V_{OUT} = 1.2V(1 + R1/R2) \tag{6}$$

ID	Device	Voltage	Chip supply	Max current
1	ADM7172	4V	Transmitter	58mA
2	ADM7172	2.7V	Transmitter	277mA
3	ADM7172	2.7V	Receiver	300mA
4	ADP7118	1.35V	Transmitter	11mA
5	ADP7118	1.35V	Receiver	1mA
6	ADP7118	2.85V	IF Downconverter	200mA
7	AMS1117	3.3V	IF Downconverter	100mA
8	AMS1117	3.3V	Multiple	340mA

Table 4: Implemented voltage sources on the board

4.2.7 Antenna

One of the most crucial elements in a radar sensor like this one is the antenna. It allows the transmission and reception of a precise frequency signal with the required distribution and efficiency. The antennas performance primary depends on its gain which is the ratio between the energy propagated in the target direction compared to the energy propagated by an identical isotropic antenna. Type, size, shape and orientation of the antenna characterize the design and effectiveness of a system. These properties are determined according to the carrier RF frequency as well as the desired radiation pattern.

For this Doppler radar system, the antenna for transmitting and receiving 60GHz millimeterwave signals is a 4x1 series fed rectangular patch array antenna. Every element has the same dimensions based on the antennas optimal performance. The patch elements are connected using microstrip lines. Transmitter antenna and receiver antenna are integrated on the DRS board, positioned parallel and opposite from each other. The radiating patch is made up of a conducting material, in this case gold. A specified dielectric substrate separates the antenna layer with the ground plane of the PCB. Shown in Fig. 26 are the dimensions of the 4-element patch array designed by Dr. Ma Chao for this specific sensor.

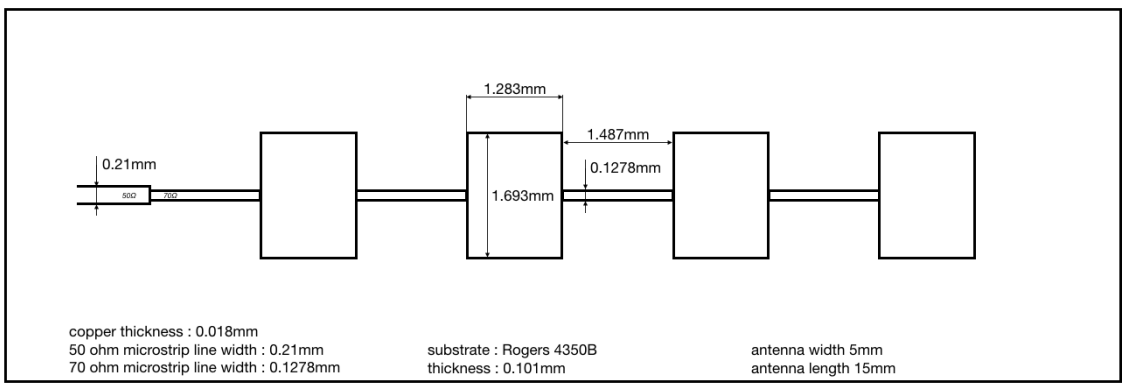


Figure 26: Antenna design dimensions

4.2.8 Power consumption

The DRS board is power supplied by an external portable battery, usually used to charge smartphones, it is capable of delivering 5V and a current of up to 2A. In order to reduce the size and weight of this battery, the power consumption of each chip should be implemented as low as possible without affecting the sensor performance. Based on the datasheets in the appendices, a list of maximum power consumption for every IC on the PCB is created and represented in Table 5.

ID	Device	Max power
1	AD9550	450 mW
2	AD9833	12.65 mW
3	ADM7172	43.5 mW
4	ADP7118	1.6 mW
5	AMS1117	55 mW
6	CP2102	130 mW
7	HC05	15 mW
8	HMC6300	992 mW
9	HMC6301	820 mW
10	MAX2831	900 mW
11	SiT5000	109 mW
11	STM8L	264 mW
	Total	3.79 W

Table 5: Maximum power consumption on the board

4.3 PCB

4.3.1 Schematics

All the components placed on the PCB board are connected together as seen previously, using Cadence to create the schematics. Passive components, i.e. resistors and capacitors, are implemented as external filters, voltage dividers and decoupling capacitors. The circuitry and value of these components are based on equations seen previously (6) and schematics from existing evaluation boards.

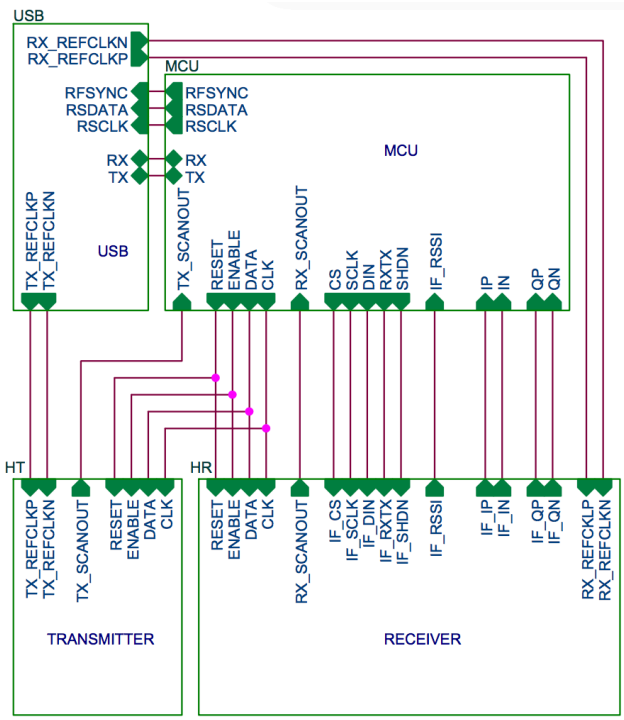


Figure 27: Schematics block structure

Fig. 27 shown above, is the block structure describing the signal distribution and connections between each part of the PCB. The PDF version of the schematics including bill of materials can be found in the corresponding appendices 4.3.1.

4.3.2 Layout

The printed circuit board consists of 4 layers and is minimized to a size of 58x50mm. Table 6 describes the layout cross section where thickness is given in mm. Every layer is available in the layout file in Appendix 4.3.2. Via arrays, connected to the ground, are positioned under potential heating chips to create thermal dissipation, i.e. the RF transceiver MAX2831, the clock translator AD9550 and the USB bridge CP2102. Decoupling capacitors are placed as close as possible to each supply pin to achieve their best performance.

ID	Layer	Material	Thickness
1	Top	copper	0.03048
2	Substrate	FR-4	0.2032
3	Power	copper	0.03048
4	Substrate	FR-4	0.2032
5	GND	copper	0.03048
6	Substrate	RO4350B	0.101
7	Bottom	copper	0.018

Table 6: Layout cross section

Top Layer The STM8L microcontroller with its crystal is placed on the top layer along with all the voltage regulators powered from the USB parts. Additionally, two J4 connectors are arranged on this layer for the SWIM communication and the Bluetooth module. A net plane ensures proper ground connection for every component.

Intermediate Layers A power plane distributes the corresponding voltage supplies to every part on both sides of the board. On this plane are also a few signal lines, i.e. SPI lines for transmitter and receiver. The second intermediate layer is used as the ground plane, located right before the bottom layer to assure isolation between the radiating antenna patch arrays and other chips.

Bottom Layer All the RF components are placed on the bottom layer, i.e. both transmitter and receiver chips with their antennas placed opposite from each other as well as the ICs in charge of generating the reference clock. In this way, no high frequency lines have to go through vias and are as short as possible in order to avoid signal disturbances.

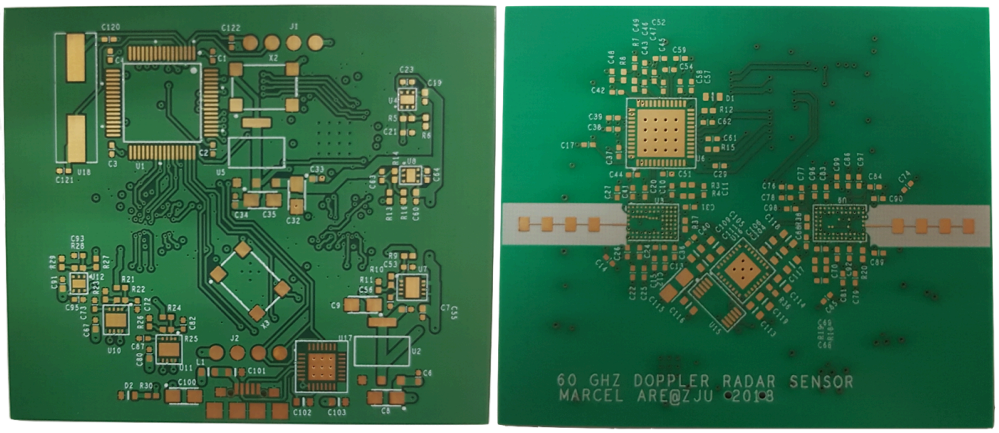


Figure 28: PCB top layer (left) and bottom layer (right)

5 Software

5.1 MCU Software

The microcontroller software principal function is to control the ICs on the board in order to ensure the correct frequency and downconversion settings. Then it retrieves and converts the sensor information to transfer the data for further signal processing. The entire code of the software is located as appendix 5.1.1 to this document.

5.1.1 Software Architecture

The flash memory of the STM8 microcontroller is loaded with the software. A C program is used in order to send and receive the necessary information through predefined ports. The main class calls external functions from hardware and hardware abstraction layer (HAL) to initialize the MCU and to program the ICs registers by using SPI simulations. In the while loop, ADC measurements and data transfer is executed. Figure 29 is a package diagram showing how each class is connected and called from the main layer.

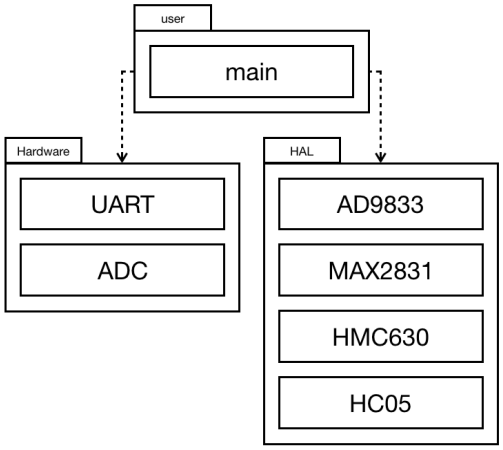


Figure 29: Software architecture package diagram

5.1.2 Register programming

At launch of the code, the hardware components of the MCU used in this software (shown in Figure 29) are initialized and set to operate for the desired applications. After this, the chips located on the board are configured to their specific settings. For each device, a 3-wire SPI Interface is simulated on the corresponding ports in order to load data bits into the programmable registers. These data arrays are chosen based on multiple datasheets (see sources and appendices) where the instructions of each register array and the SPI timing diagrams are described. A table in appendix 5.1.2 lists all the determined register arrays for each programmed component. The proper functioning of this implemented SPI simulation code is verified in the debug report also located at the end of this document.

5.1.3 Data conversion

Received and downconverted differential I/Q signals input the microcontroller over the ADC pins channel 1 to channel 4. In order to sample these four signals conveniently, single mode is used to convert each channel separately and store the data on a rolled buffer. The sampling time for an ADC channel is determined by a number of clock cycles, in this case 48 corresponding to approx. 3 ms. After the conversion of all four channels, the extracted data stored in the buffer is sent via UART to the Bluetooth module and micro USB port. Figure 30 below demonstrates how the I/Q signals are sampled and converted.

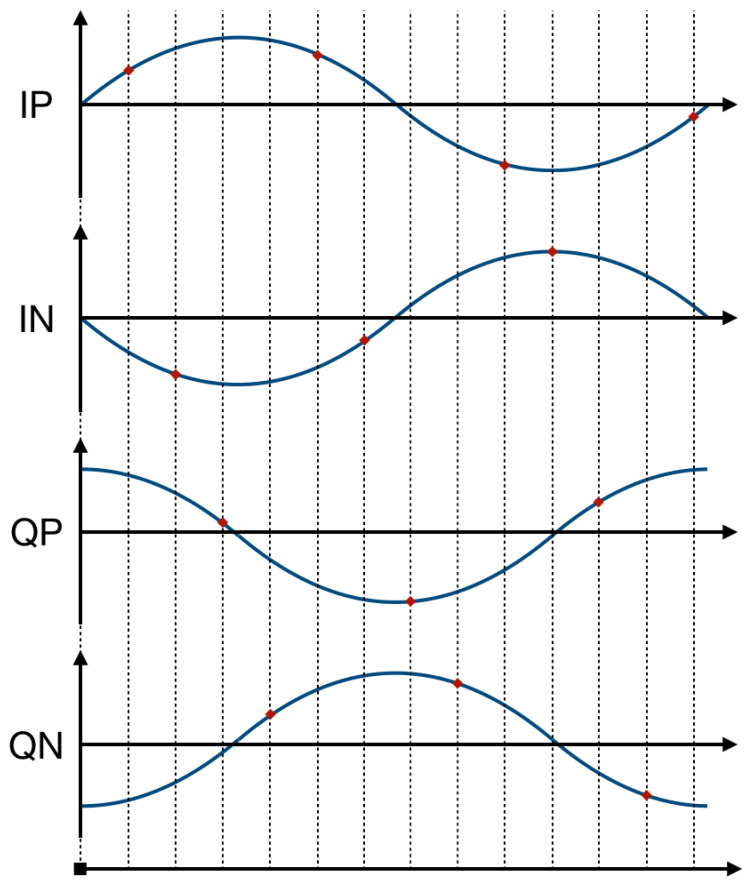


Figure 30: I/Q signals ADC sampling

Bluetooth or USB transmission transfers the conversion data to a DSP software on PC. Further data processing allows linearized, distance-independent Doppler phase demodulation as well as respiration and heartbeat spectra retrieving by introducing algorithms and filters. The DSP software is collected from previous vital sign detection projects.

6 Experiments

6.1 PCB Debug

This part summarizes the debugging process i.e. the problems encountered and the solutions applied on hardware and software. The DRS PCB debug protocol, situated in appendix 5.1.2, describes each step conducted in order to assure the proper functioning of the VSD sensor.

6.1.1 Hardware

On the data port of the IF downconverter no output could be measured. The microcontroller pin on which this signal was assigned to had a I2C1 data function and could not be configured as an push-pull output. In order to send data bits to the MAX2831, the port connection had to be modified on the board layout. As a temporary solution, a wire is placed to the newly assigned pin. The software is edited to match with the hardware implementation.

On the output of the programmable waveform generator no frequency is detected. A wrongly implemented capacitor was creating a low pass filter, thus cutting off the desired output frequency. By removing this capacitor from the board, the waveform is generated correctly without causing further issues.

The voltage amplitude ranges of the waveform generator output and the frequency translator input were not correlating, causing no sine reference clock for both transmitter and receiver. A voltage amplifier is implemented and placed between the AD9833 and AD9550 to increase the waveform amplitude.

Another hardware problem prevented the clock translator to output the correct frequency. Unfortunately no solution was found in time to fix this issue.

6.1.2 Software

Apart from the pin change, no other modifications had to be applied to the software for now.

7 Conclusion

7.1 Work summary

Final product The work presented at the end of this project consists of a final PCB design including all the components necessary to implement a functional miniaturized continuous-wave Doppler radar sensor for human non-contact vital sign detection. The on-board microcontroller is loaded with a simple software capable of configuring the sensor to its proper characteristics and specifications. It also controls retrieving, conversion in digital domain and further transmission of the Doppler shift data in order to achieve accurate measurements.

Summary First, the Doppler effect and its use in velocity censoring has been studied to understand the principal mechanism of this type of radar system. With these information, a general block diagram is drawn along a brief design description of the DRS. Furthermore, objectives and tasks have been determined to detail and organize the work more accurately. By looking carefully at existing Doppler sensors and research efforts about this matter, the architecture and carrier frequency ideally adapted for vital sign detection is chosen. After setting these proper specifications of the project, the components used on the PCB are chosen correspondingly and the schematics as well as layout are drawn. The MCU software is written based on previous projects using the same microcontroller family. The proper functioning of every part in this project has been tested and encountered problems are partially solved.

Structure The carrier frequency in the 60GHz ISM-band allows long-range detection and greater sensitivity to small movements of the target. This feature is important as heartbeat and respiration displacements only covers a few millimeters. Additionally, to avoid flicker noise and null point issues, an low-IF architecture is adapted. In order to use the microcontroller integrated ADCs, an intermediate frequency of 1kHz is chosen. With this method the I/Q signals, containing the Doppler shift information, can be sampled, converted and reconstructed in data processing.

Objectives The electronic and mechanical objectives fixed at the beginning of the project have been fulfilled. By using the STM8L and other low-power components, the power consumption of the system is kept as low as possible so that the system can be supplied by a single small, longer lasting battery connected to the micro USB port. The entire radar sensor is packed on a 4 layers PCB board of 58x50mm dimensions including a pair of patch array antennas. Its miniaturized size allows the DRS to be placed discreetly and according to its best performance. As for the data processing objectives, no experiments can confirm the precision of the existing algorithms. However by observing the results from other similar projects using the same data processing methods [7], it can be concluded that, with specific modifications of the algorithms and filters, the measurements for this sensor would be accurate enough to properly identify human vital signs.

Tasks All the main objectives and tasks have been accomplished. At the end, a few steps were missing for the finalization of the project. These tasks will be completed in a future project. Additionally, the board can be used for various other purposes. This is further discussed in the next section.

7.2 Future tasks

As there was not enough time left to finish debugging the PCB and software entirely, these tasks will be reported to the continuation of the project. Once the board is fully debugged and properly functioning, existing algorithms and filters can be used to extract heartbeat and respiration rate from the converted data provided by Bluetooth transfer. The value of the intermediate frequency can be adjusted to work ideally with the signal conversion by the microcontroller. Finally, various tests and comparisons can then be executed to demonstrate accurate non-contact detection of human vital signs.

Alternatively, this board can be used for other applications as the carrier frequency is wide in range and permits the detection of targets with limited movements. The intermediate frequency can be adapted to suit perfectly the estimated Doppler shift of the moving object. A future project using this continuous wave Doppler radar is the classification and detection of unmanned aerial vehicles (UAVs), i.e. distinguishing drones from birds, planes or helicopters as considered in [8]. Furthermore the UAVs physical specifications can be measured using micro-Doppler signature as seen in [9] where number of blades, blade length and rotations per second is automatically estimated by an algorithm.

Date and Signature

Marcel Balle

_____

Signature

_____

Date

References

[1] WOLFF Christian. Radar basics: Classification of radar sets, radar frequency bands. *Radartutorial.eu*, 2014.

[2] GHOSE Debasish. Navigation, guidance and control. *NPTEL COURSE*, pages 30–33, 2014.

[3] Changzhan Gu, Ruijiang Li, Hualiang Zhang, Albert YC Fung, Carlos Torres, Steve B Jiang, and Changzhi Li. Accurate respiration measurement using dc-coupled continuous-wave radar sensor for motion-adaptive cancer radiotherapy. *IEEE Transactions on biomedical engineering*, 59(11):3117–3123, 2012.

[4] Travis Hall, Nicholaus A Malone, Jerry Tsay, Jerry Lopez, T Nguyen, Ron E Banister, and DY C Lie. Long-term vital sign measurement using a non-contact vital sign sensor inside an office cubicle setting. In *Engineering in Medicine and Biology Society (EMBC), 2016 IEEE 38th Annual International Conference of the*, pages 4845–4848. IEEE, 2016.

[5] MS Rabbani and H Ghafouri-Shiraz. 60 ghz microstrip antenna for remote vital sign monitoring in automobile applications. 2017.

[6] L Pham and PWC Prasad. An exploration of non-contact cardiopulmonary measurement using the smartphone in rescue relief events. In *Advances in Electrical, Electronic and Systems Engineering (ICAEES), International Conference on*, pages 258–263. IEEE, 2016.

[7] Qinyi Lv, Lei Chen, Kang An, Jun Wang, Huan Li, Dexin Ye, Jiangtao Huangfu, Changzhi Li, and Lixin Ran. Doppler vital signs detection in the presence of large-scale random body movements. *IEEE Transactions on Microwave Theory and Techniques*, 66(9):4261–4270, 2018.

[8] Pavlo Molchanov, Ronny IA Harmanny, Jaco JM de Wit, Karen Egiazarian, and Jaakko Astola. Classification of small uavs and birds by micro-doppler signatures. *International Journal of Microwave and Wireless Technologies*, 6(3-4):435–444, 2014.

[9] Ashish Kumar Singh and Yong-Hoon Kim. Automatic measurement of blade length and rotation rate of drone using w-band micro-doppler radar. *IEEE Sensors Journal*, 18(5):1895–1902, 2018.

Sources

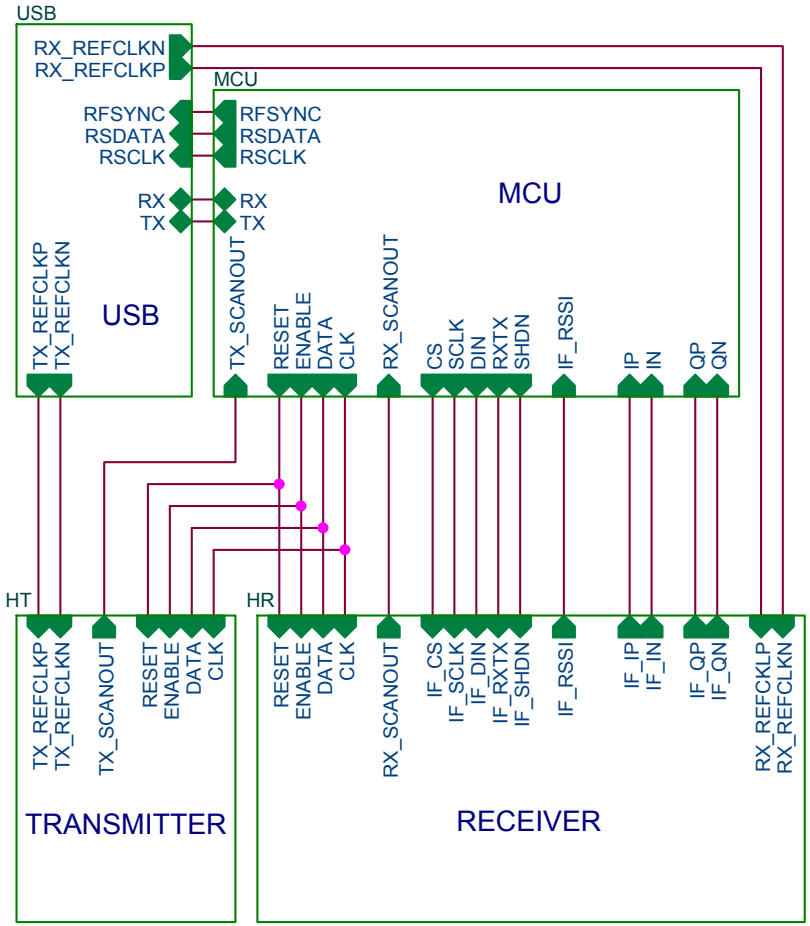
- Stoehr, Martin D., and ISMRF PMTS. "RF Basics." PMTS, ISM-RF Strategic Applications.
- Ghose, Debasish. "Navigation, Guidance and Control." NPTEL COURSE (2014).
- Mirabbasi, Shahriar, and Ken Martin. "Classical and modern receiver architectures." IEEE Communications Magazine 38.11 (2000): 132-139.
- Cruz, Pedro, Hugo Gomes, and Nuno Carvalho. "Receiver Front-End Architecturesanalysis and Evaluation." Advanced Microwave and Millimeter Wave Technologies Semiconductor Devices Circuits and Systems. InTech, 2010.
- Schematics HMC6350EK
- Datasheet MAX2831 Evaluation Kit
- Datasheet AD9833 Evaluation Board User Guide
- Datasheet Programming the AD9833
- Schematics STM8L Discovery
- Reference manual STM8
- Application note ADC STM8L

Appendices

Components

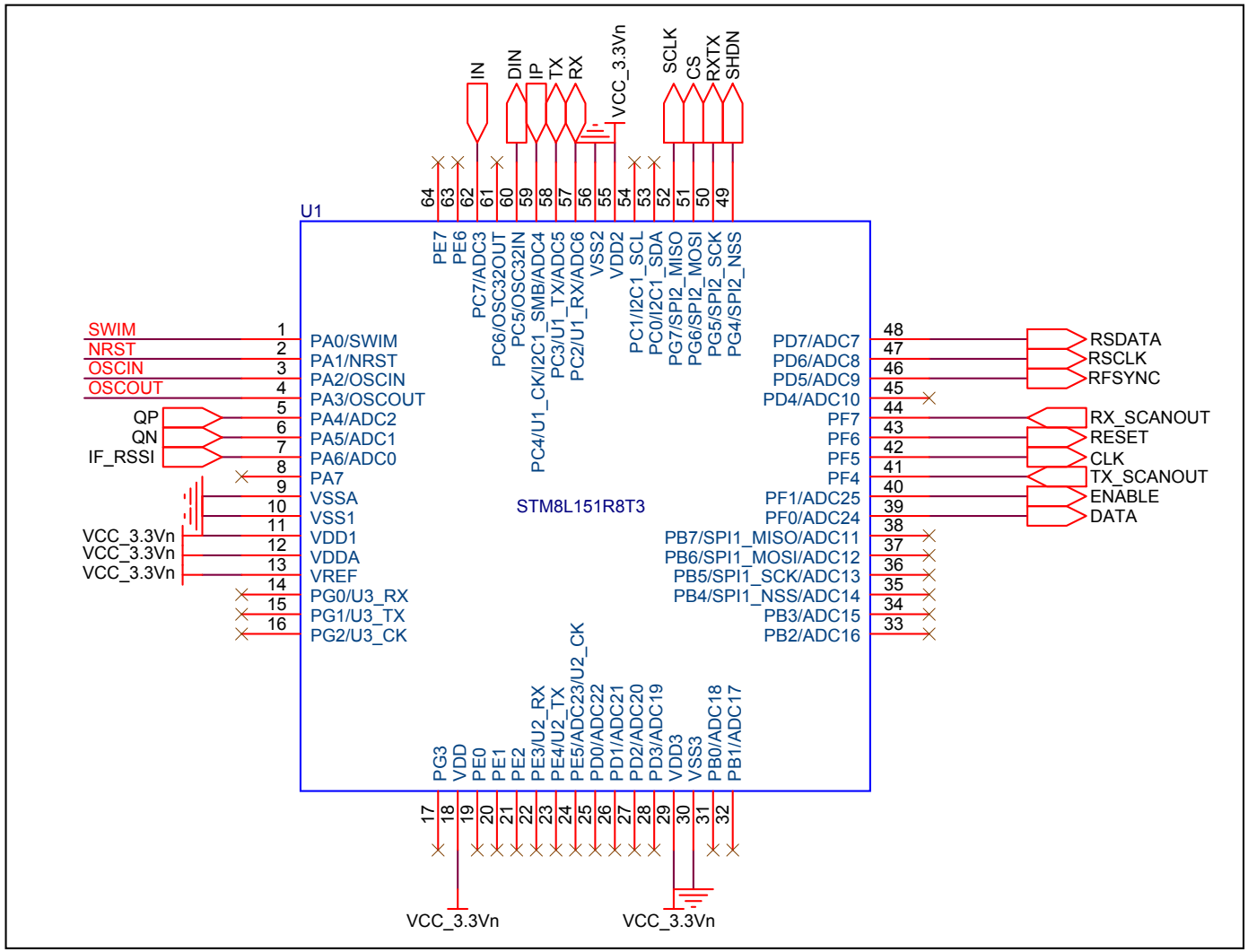
PCB

Software

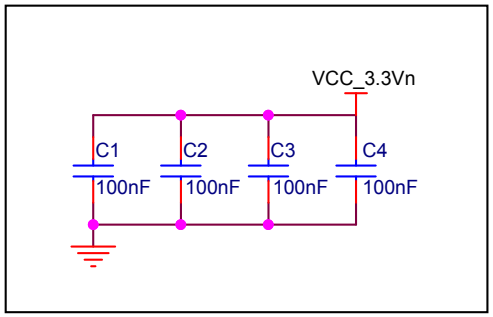


Title	SENSOR BLOCK	
Size A	Document Number 000	Rev 0
Date: Thursday, August 09, 2018	Sheet 1 of 5	

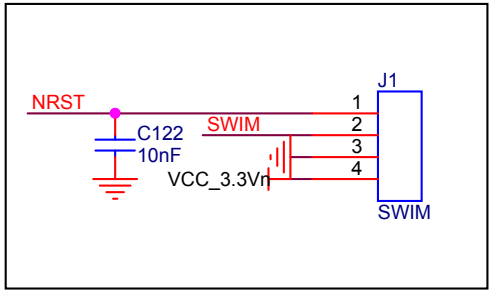
Microcontroller



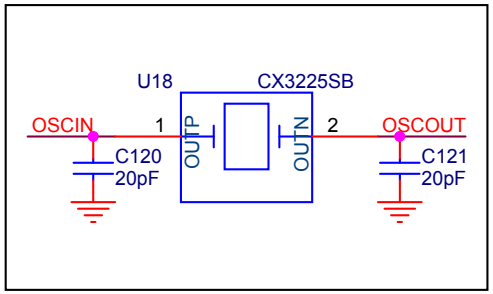
Decoupling capacitors



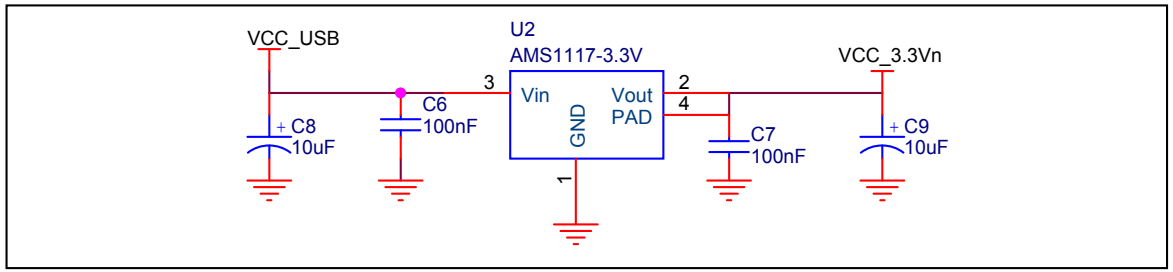
Programming Connector



Crystal oscillator



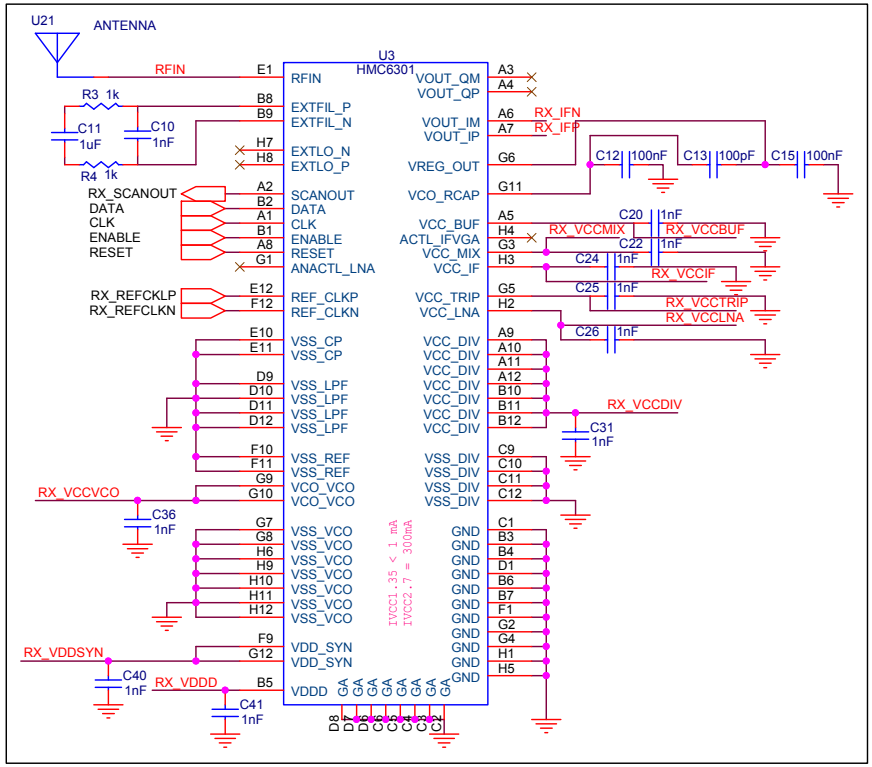
MCU voltage regulator



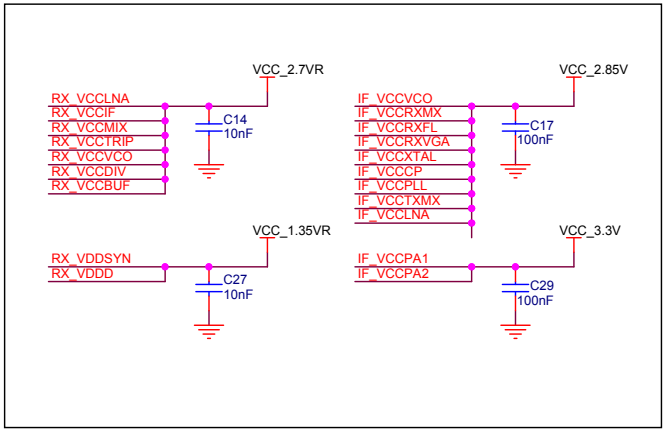
VCC_USB = 5V

Title	SENSOR MCU	
Size A	Document Number 001	Rev 0
Date: Monday, August 27, 2018	Sheet 2 of 5	

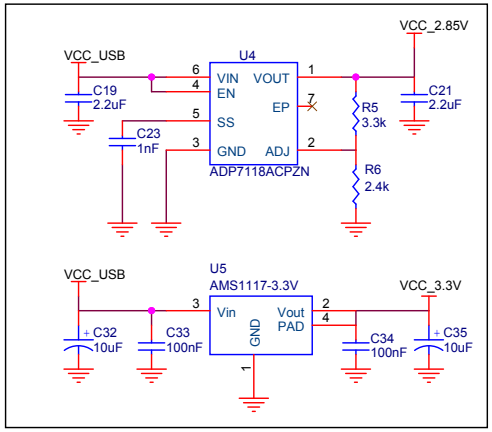
Receiver



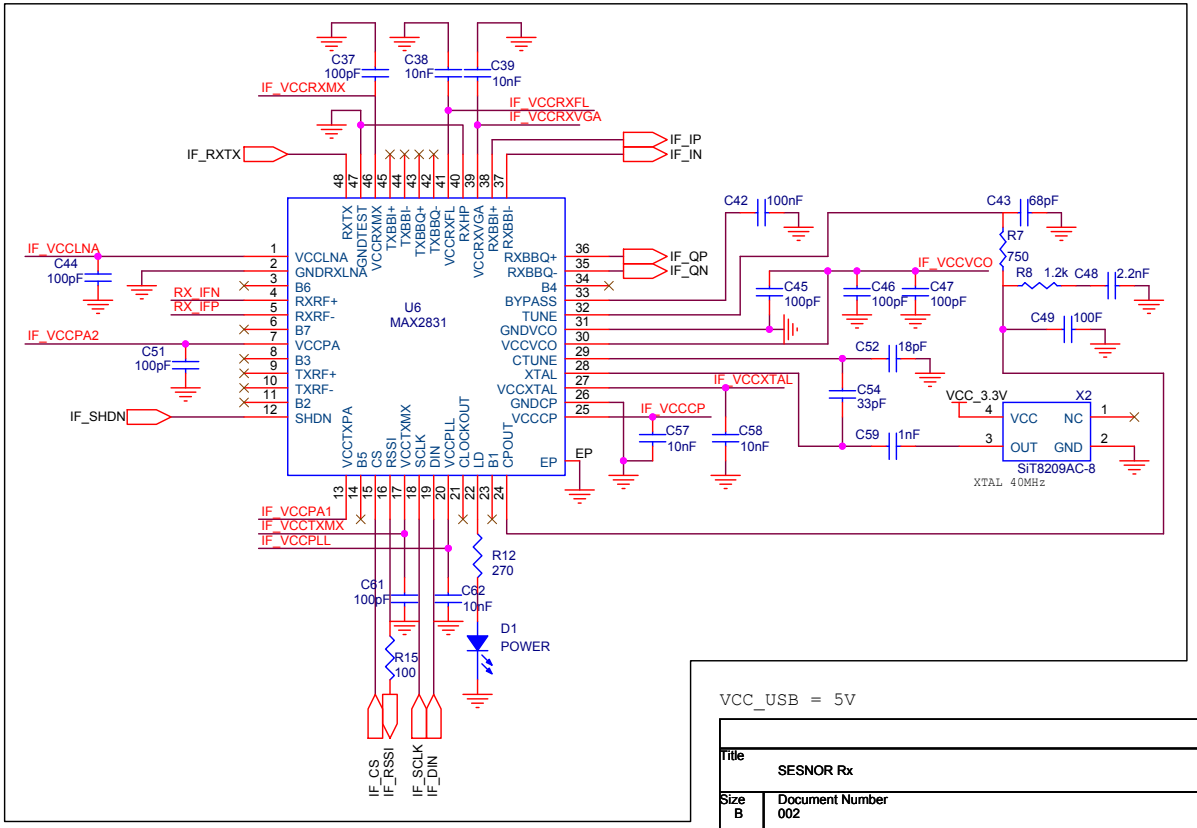
Supply distribution



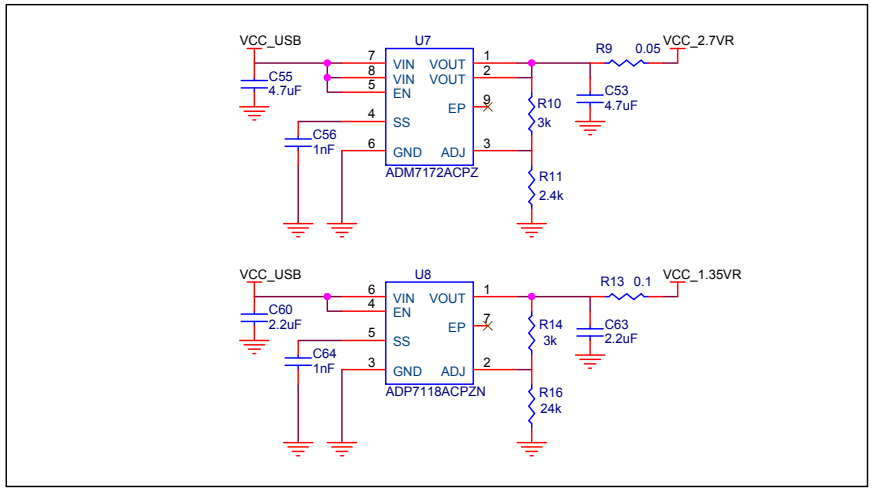
General voltage regulators



IF Downconverter



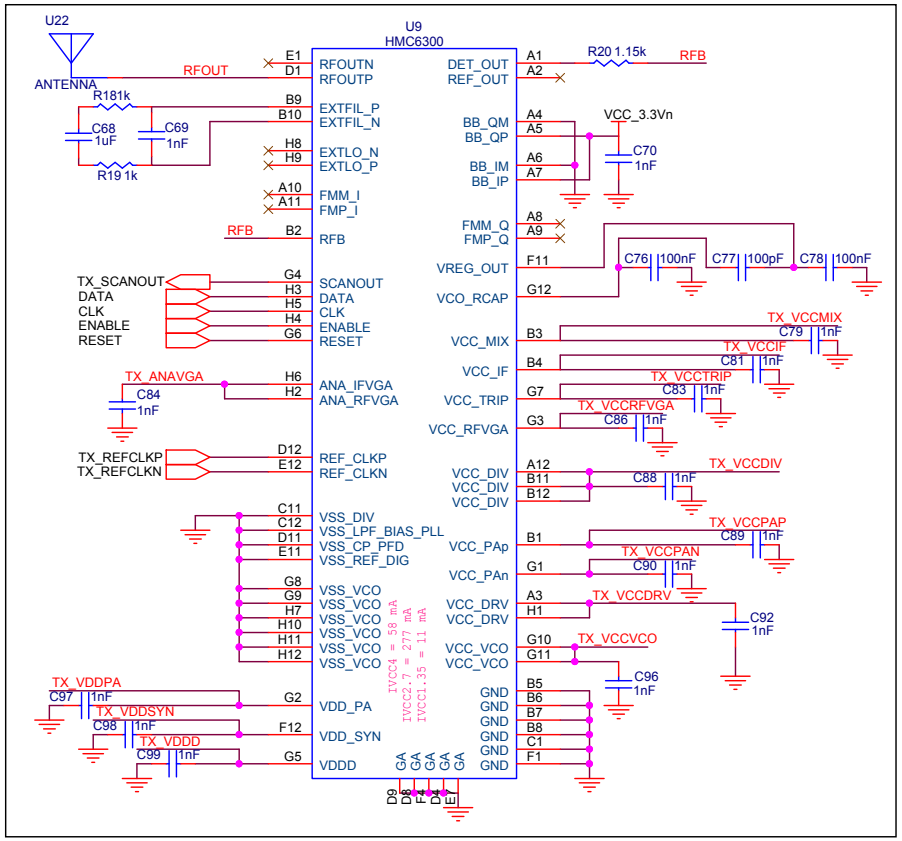
Receiver voltage regulators



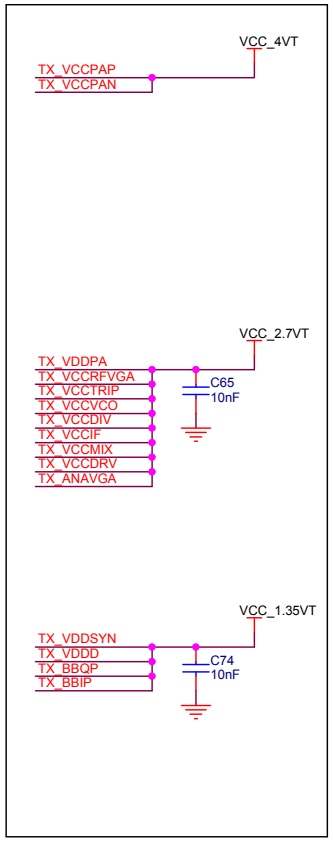
VCC_USB = 5V

Title	SESNOR Rx		
Size B	Document Number 002		Rev 0
Date: Monday, August 27, 2018		Sheet 3 of 5	

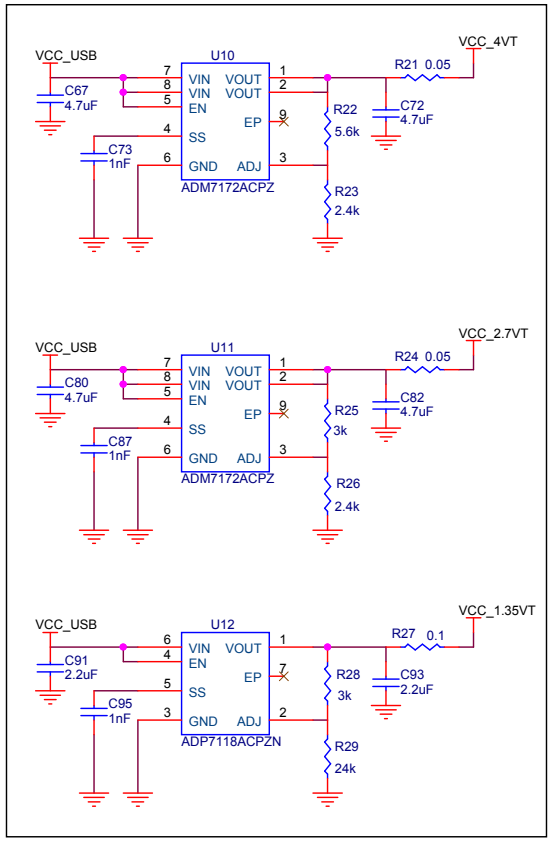
Transmitter



Supply distribution



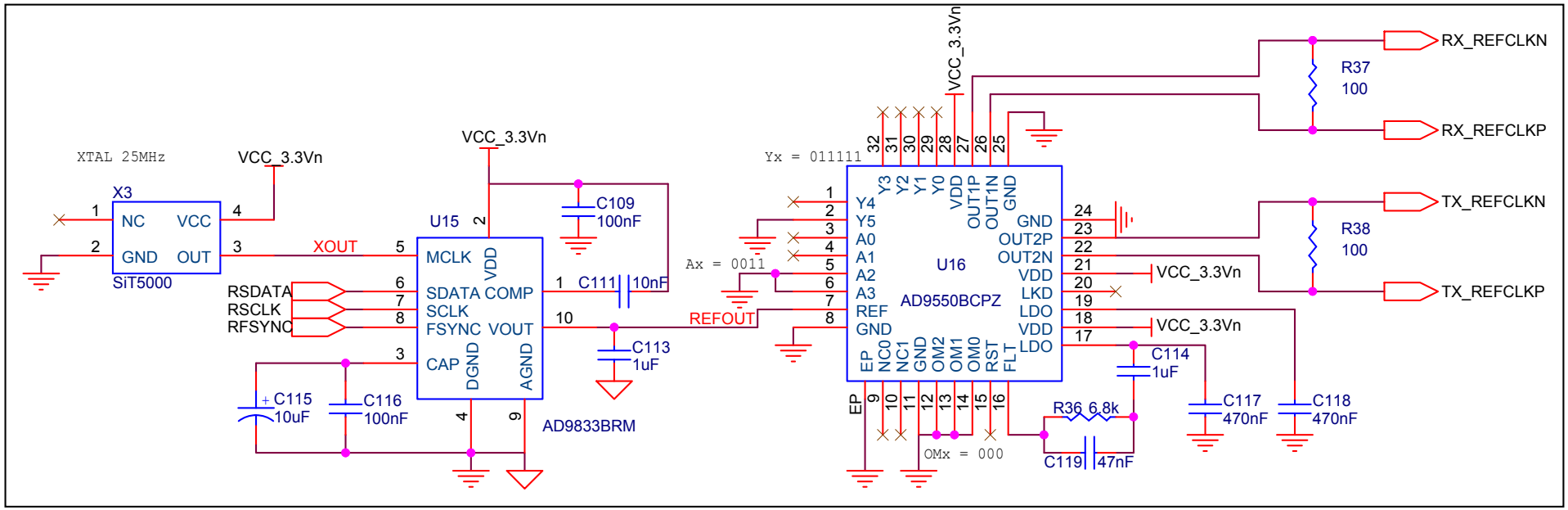
Transmitter voltage regulators



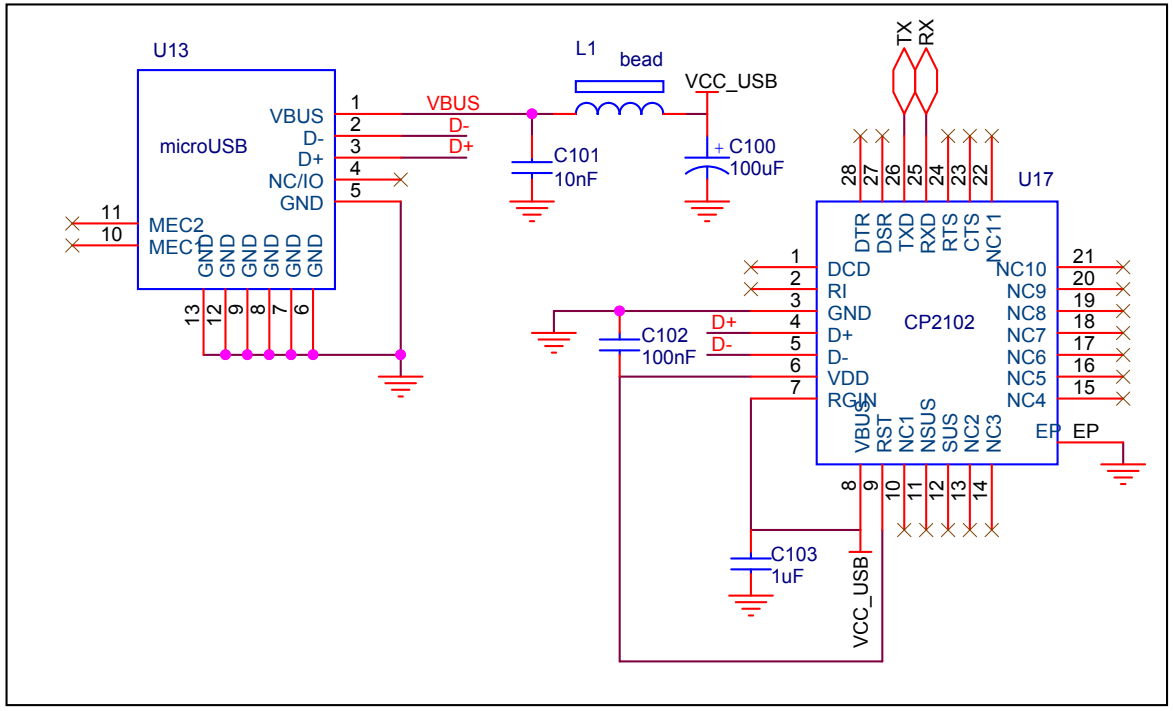
VCC_USB = 5V

Title	SENSOR Tx	
Size B	Document Number 003	Rev 0
Date: Tuesday, August 21, 2018	Sheet 4 of 5	

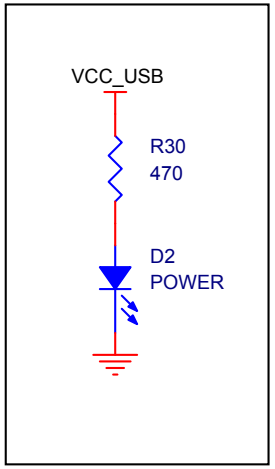
Reference clock generator



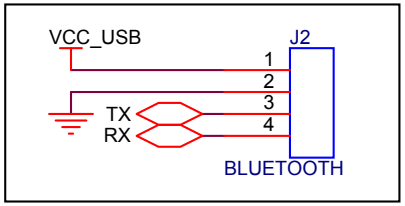
USB interface



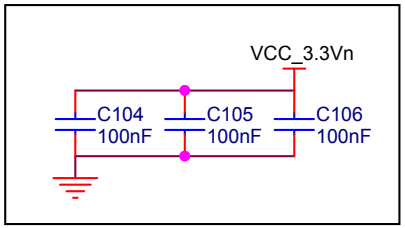
Supply LED



Bluetooth module



Decoupling capacitors

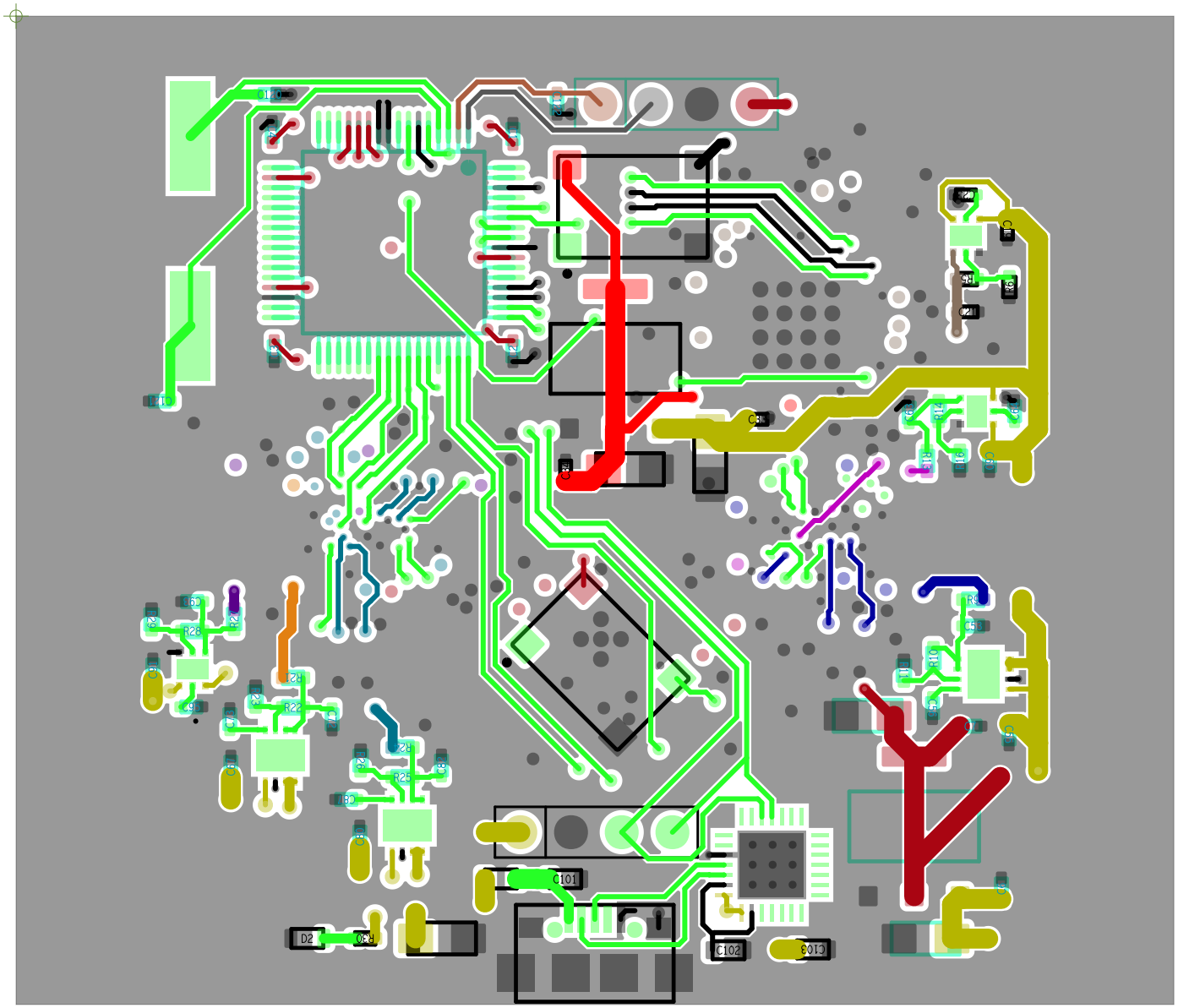


VCC_USB = 5V

Title	SESNOR USB	
Size A	Document Number 004	Rev 0
Date: Tuesday, August 21, 2018	Sheet 5 of 5	

Bill Of Materials			
Item	Quantity	Reference	Part
1	21	C1,C2,C3,C4,C6,C7,C12,C15,C17,C29,C33,C34,C42,C76,C78,C102,C104,C105,C106,C109,C116	100nF
2	5	C8,C9,C32,C35,C115	10uF
3	32	C10,C20,C22,C23,C24,C25,C26,C31,C36,C40,C41,C56,C59,C64,C69,C70,C73,C79,C81,C83,C84, C86,C87,C88,C89,C90,C92,C95,C96,C97,C98,C99	1nF
4	5	C11,C68,C103,C113,C114	1uF
5	9	C13,C37,C44,C45,C46,C47,C51,C61,C77	100pF
6	12	C14,C27,C38,C39,C57,C58,C62,C65,C74,C101,C111,C122	10nF
7	6	C19,C21,C60,C63,C91,C93	2.2uF
8	1	C43	68pF
9	1	C48	2.2nF
10	1	C49	100F
11	1	C52	18pF
12	6	C53,C55,C67,C72,C80,C82	4.7uF
13	1	C54	33pF
14	1	C100	100uF
15	2	C117,C118	470nF
16	1	C119	47nF
17	2	C120,C121	20pF
18	2	D1,D2	POWER
19	1	J1	SWIM
20	1	J2	BLUETOOTH
21	1	L1	bead
22	4	R3,R4,R18,R19	1k
23	1	R5	3.3k
24	4	R6,R11,R23,R26	2.4k
25	1	R7	750
26	1	R8	1.2k
27	3	R9,R21,R24	0.05
28	4	R10,R14,R25,R28	3k
29	1	R12	270
30	2	R13,R27	0.1
31	3	R15,R37,R38	100
32	2	R16,R29	24k
33	1	R20	1.15k
34	1	R22	5.6k
35	1	R30	470
36	1	R36	6.8k
37	1	U1	STM8L151R8T3
38	2	U2,U5	AMS1117-3.3V
39	1	U3	HMC6301
40	3	U4,U8,U12	ADP7118ACPZN
41	1	U6	MAX2831
42	3	U7,U10,U11	ADM7172ACPZ
43	1	U9	HMC6300
44	1	U13	microUSB
45	1	U15	AD9833BRM
46	1	U16	AD9550BCPZ
47	1	U17	CP2102
48	1	U18	CX3225SB
49	2	U21,U22	ANTENNA
50	1	X2	SiT8209AC-8
51	1	X3	SiT5000

ART FILM - TOP

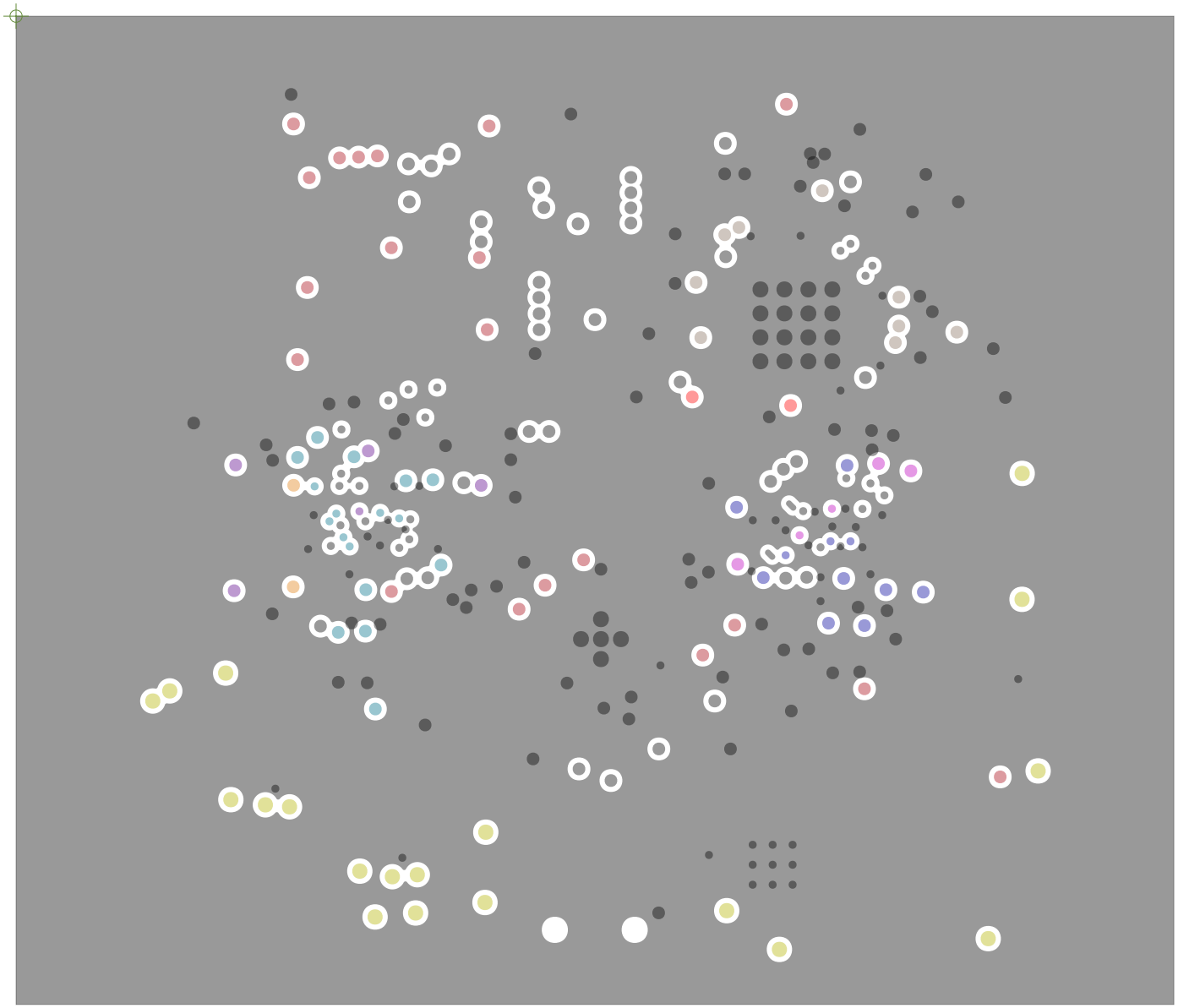


ART FILM - TOP

ART FILM - POWER

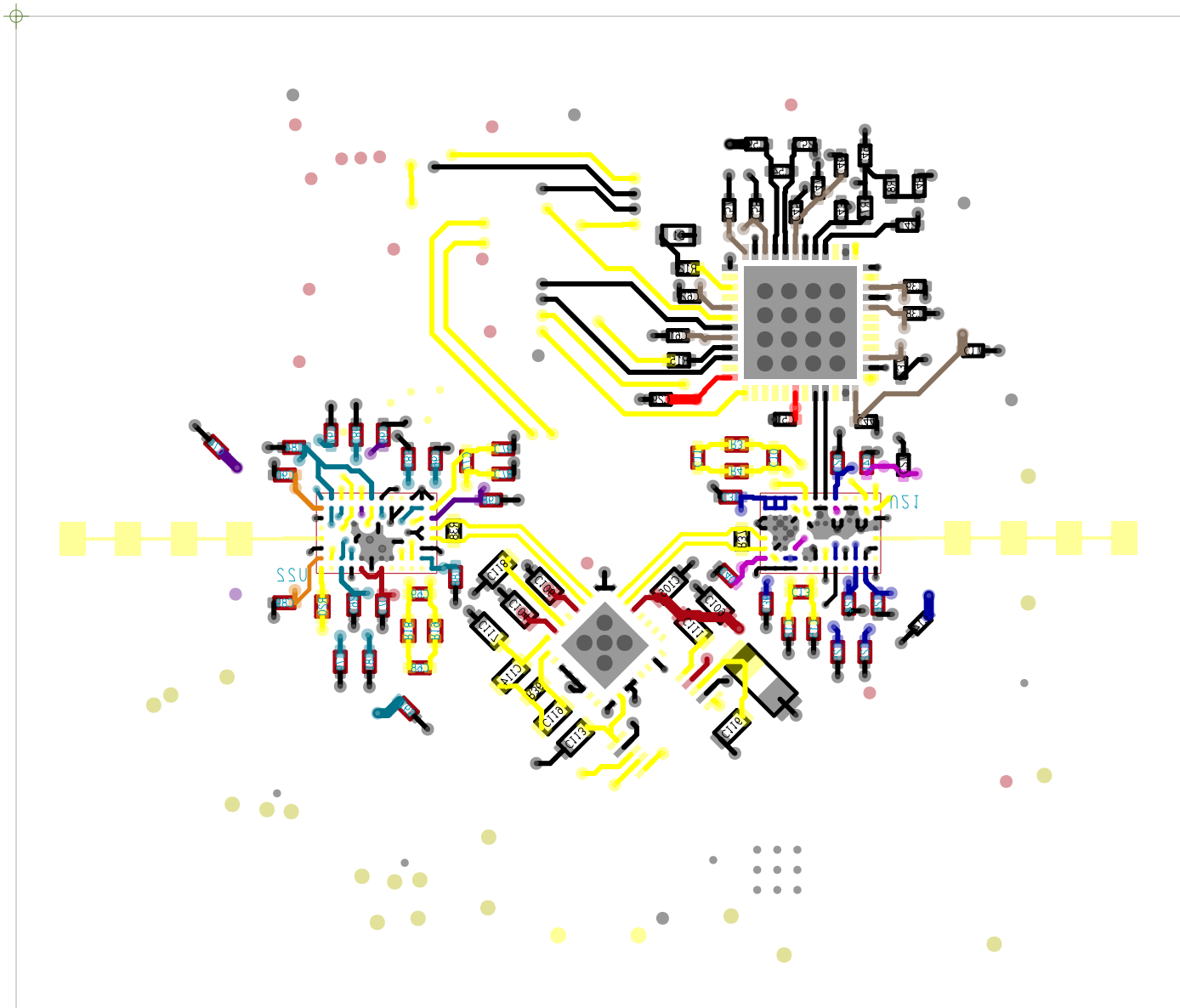
ART FILM - POWER

ART FILM - GND



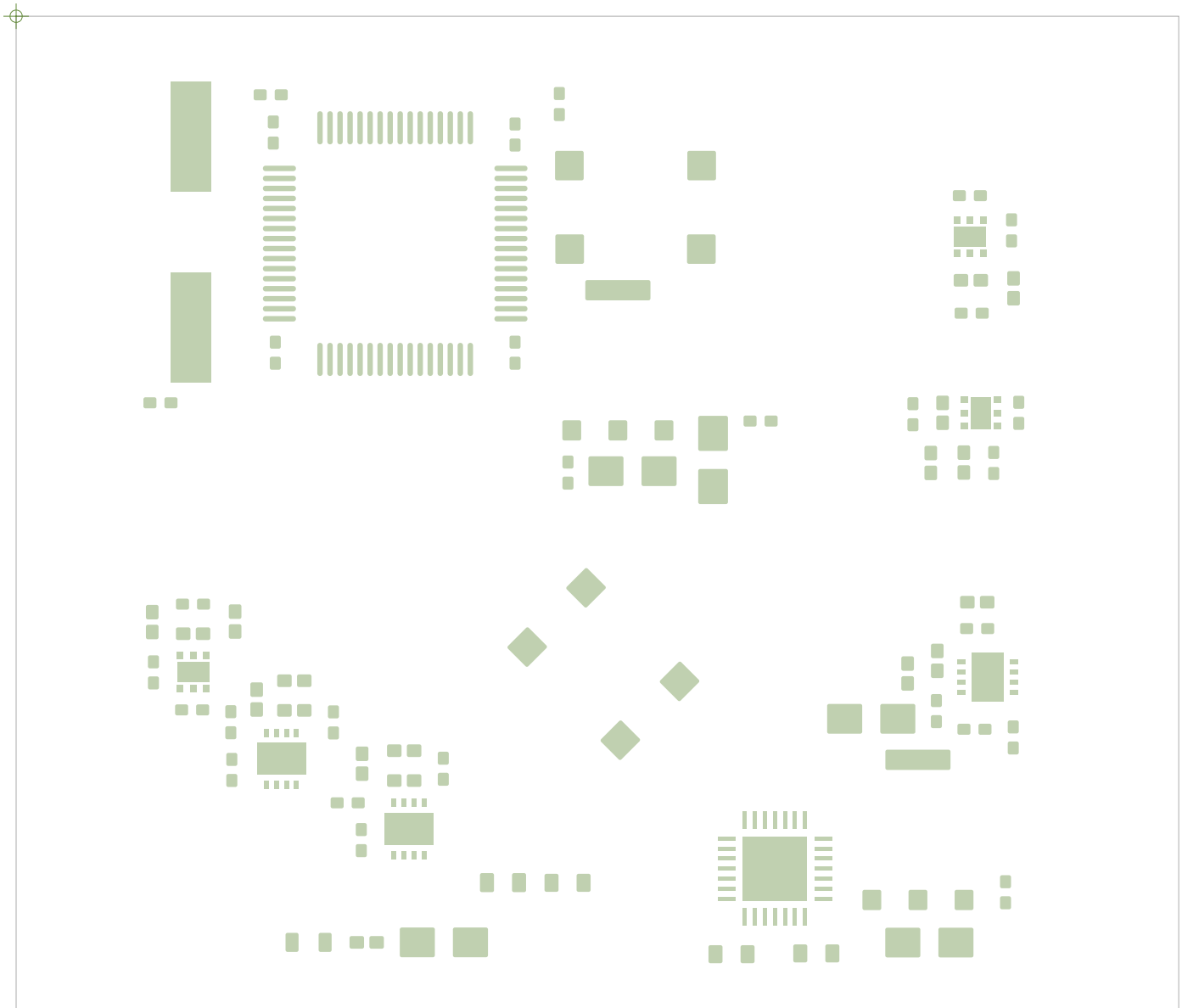
ART FILM - GND

ART FILM - BOTTOM



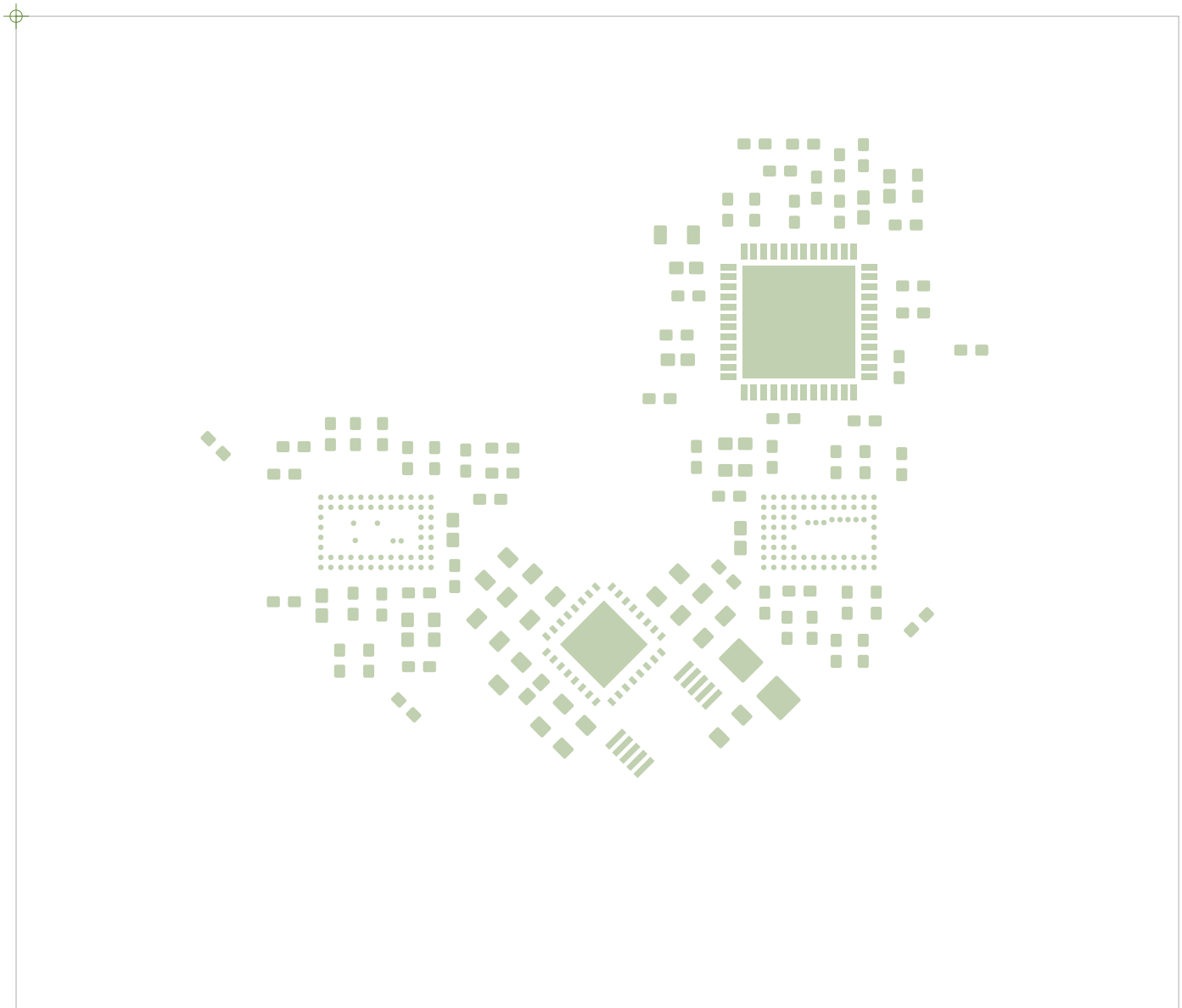
ART FILM - BOTTOM

ART FILM - PAST_TOP



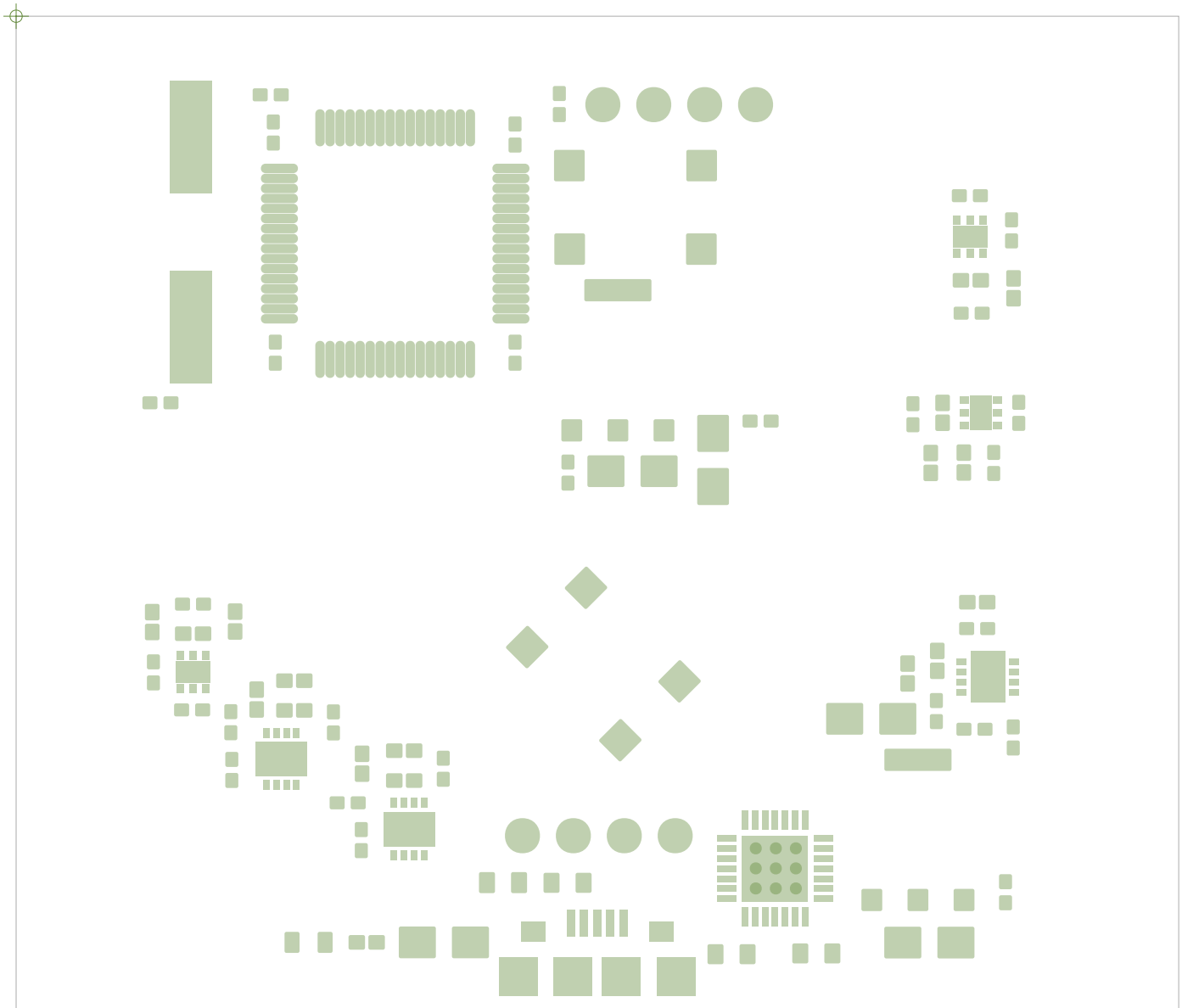
ART FILM - PAST_TOP

ART FILM - PAST_BOTTOM



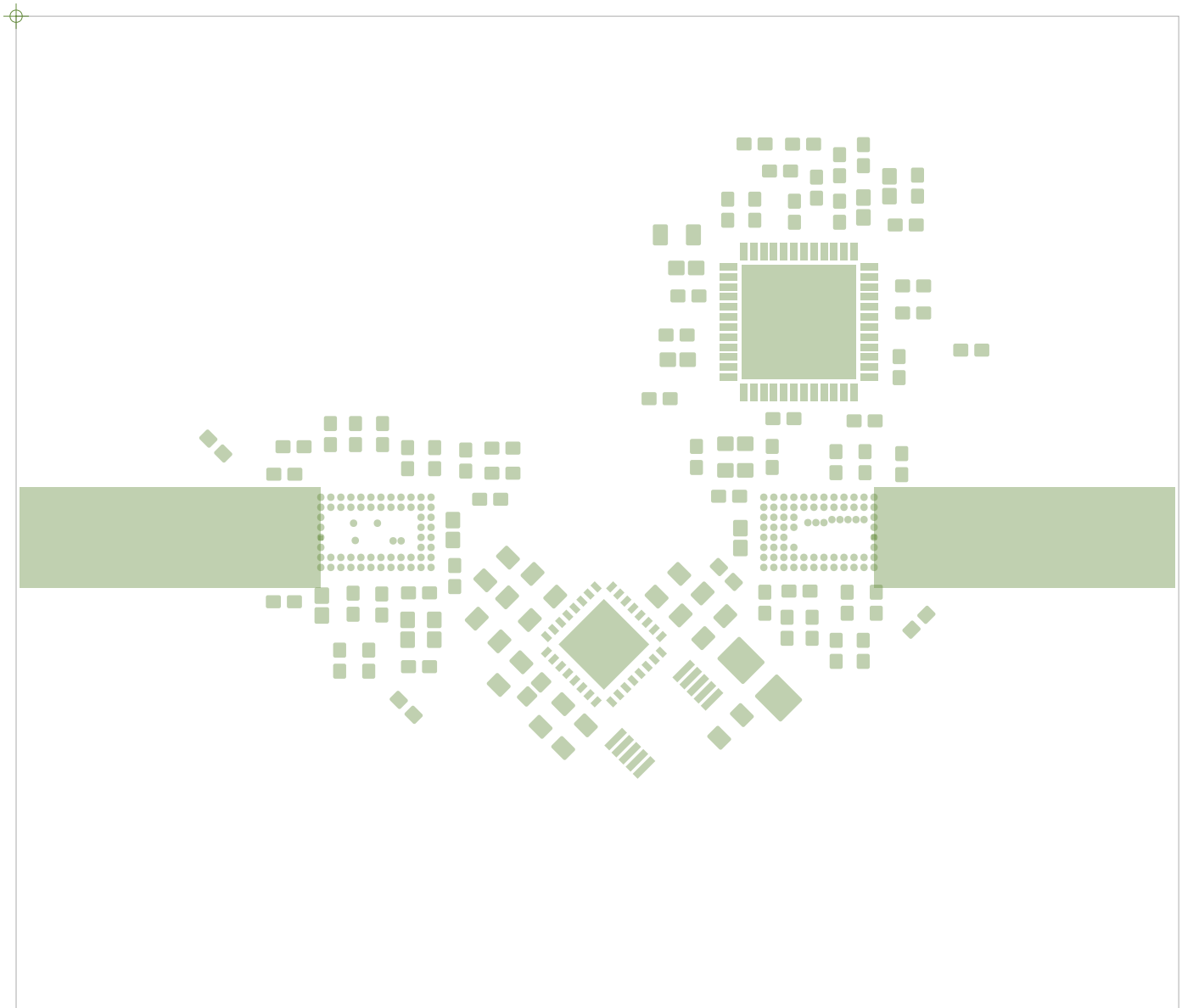
ART FILM - PAST_BOTTOM

ART FILM - SOLD_TOP

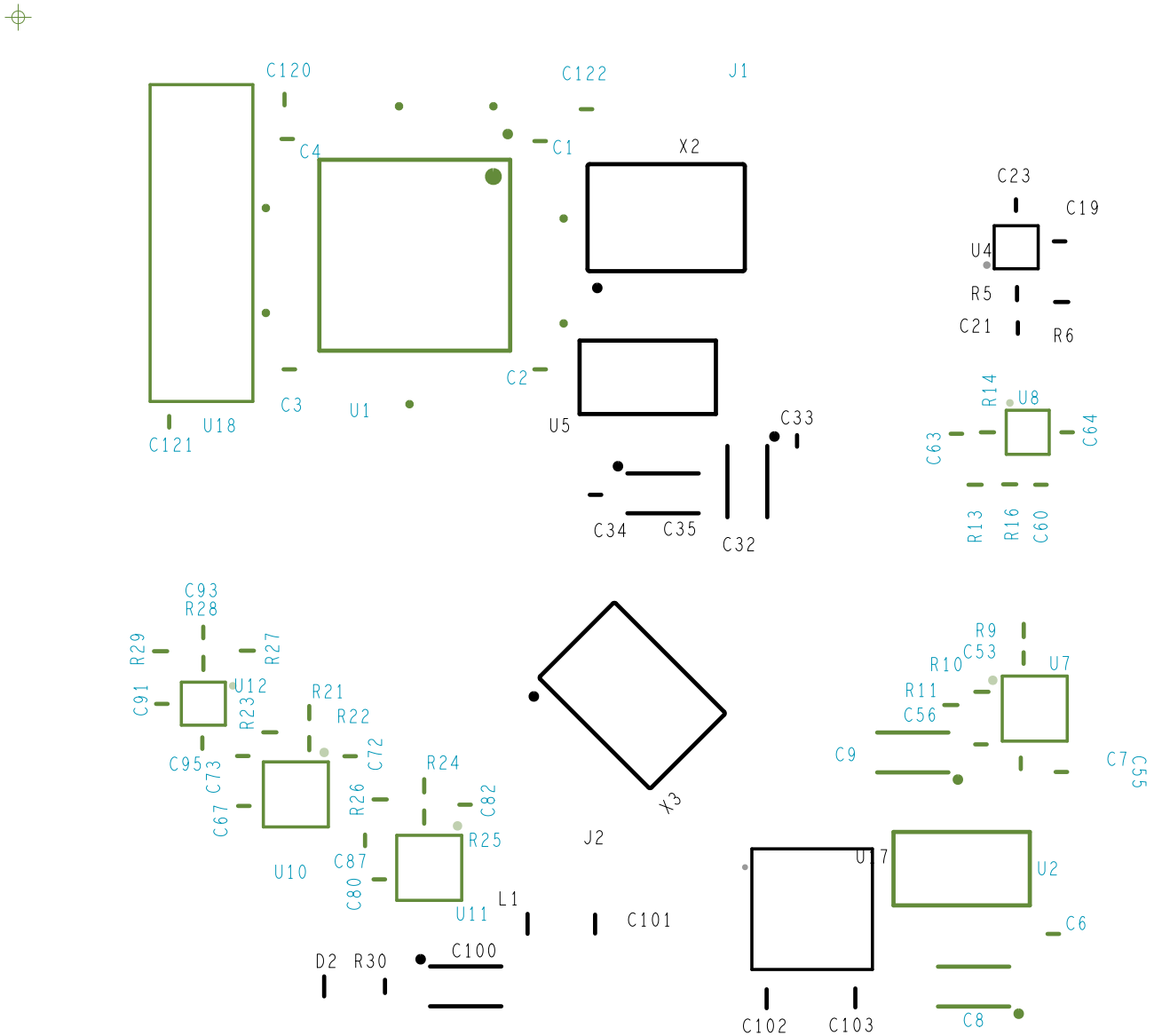


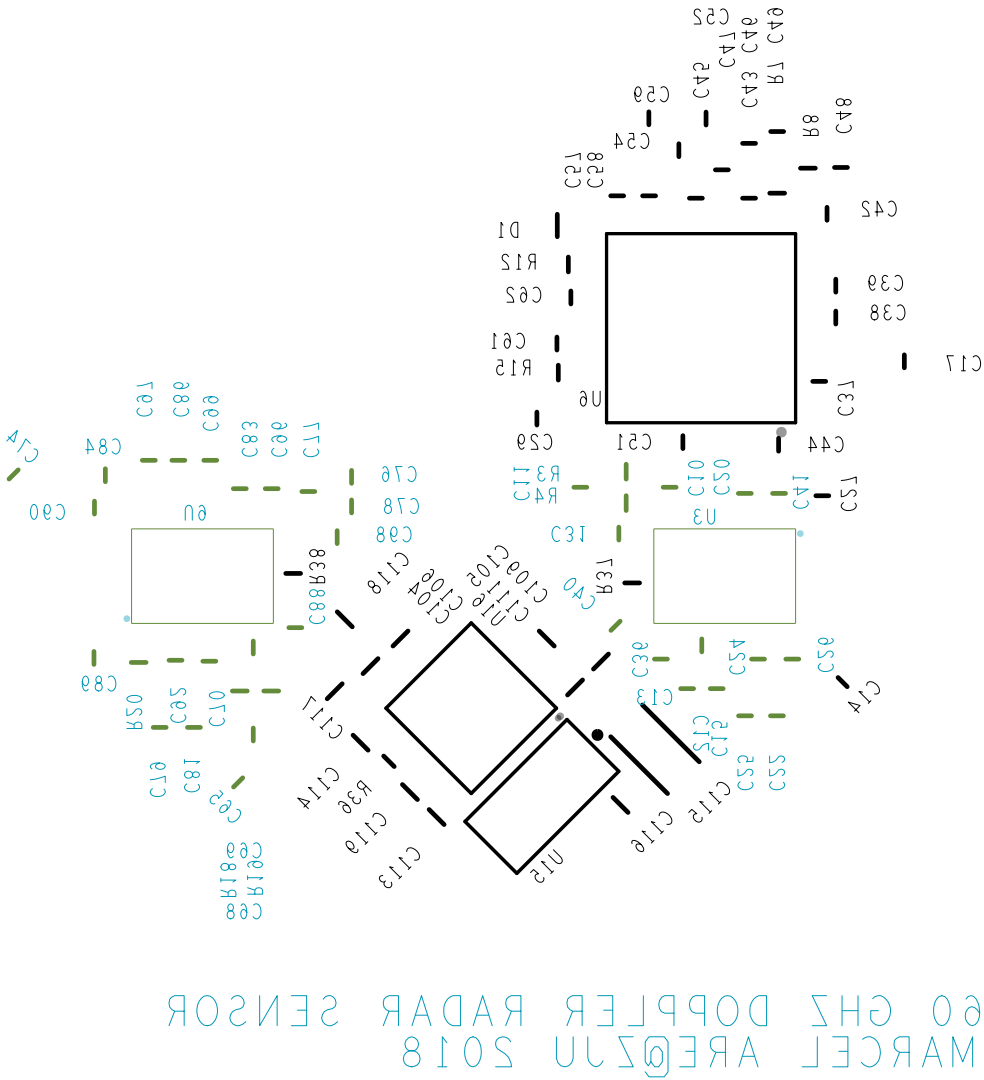
ART FILM - SOLD_TOP

ART FILM - SOLD_BOTTOM



ART FILM - SOLD_BOTTOM


C120
C122
J1
C4
C1
X2
C23
C19
U4
R5
C21
R6
C2
C3
U1
U18
C121
U5
C33
R14
U8
C63
C64
R13
R16
C60
C34
C35
C32
C93
R28
R29
R27
C91
U12
R23
R21
R22
C95
C73
C72
R24
C67
R26
C82
R25
U10
C87
C80
U11
L1
J2
X3
R9
C53
R10
U7
R11
C56
C9
C7
C55
U17
U2
C101
C6
D2
R30
C100
C102
C103
C8


60 GHZ DOPPLER RADAR SENSOR
MARCEL ARE@ZLU 2018

VCC_USB

GND

VCC_3.3Vn

VCC_3.3V

VCC_2.7VT

VCC_2.7VR

VCC_1.35VT

VCC_1.35VR

VCC_4VT

VCC_2.85V

SWIM

NRST

Signal

Signal

```c
#include "prohead.h"

#include "delay.h"
#include "adc.h"
#include "hc05.h"

#include "ad9833.h"
#include "max2831.h"
#include "hmc630.h"

#include "string.h"
#include "math.h"

u16 Buff = 1234;  // ADC buffer
u8 i, length = 0; // declare data length variable
char temp[5];     // declare 'channel data' string variable
char data[30];    // declare 'all data' string variable

main()
{
  _asm("rim");      // reset interrupts

  CLK_Init(CLK_F);  // set clock frequency

  AD9833_init();    // ref clock setup
  MAX2831_init();   // IF downconverter setup
  HMC630_init();    // Tx&Rx setup

  UART_Init();      // setup blutooth communication
  ADC_Init();       // ports & conversion mode

  while(1)
  {
    // measure on the correct channel
    ADC_On_SetChannel(1);         // select the ADC channel
    ADC_Get(&Buff);               // convert and store
    sprintf(data, "%d", Buff);    // convert num to char
    length = log10(Buff)+ 2;      // data length + ","
    strcat(data,",");             // add coma for separation

    for(i=2;i<5;i++)
    {
      ADC_On_SetChannel(i);       // select next ADC channel
      ADC_Get(&Buff);             // convert and store
      sprintf(temp, "%d", Buff);  // convert num to char
      length = length + log10(Buff)+2; // update data length
      strcat(data,temp);          // add conversion to all data
      strcat(data,",");           // add coma for separation
    }
    strcat(data,"\r\n");          // next line for next 'scan'
    // send all converted data to bluetooth module
    UART_SendStringLenght(data,(length+2));
  }
}
```

```c
#include "delay.h"

volatile u8 fac_us=0;

// initialize clock
void CLK_Init(u8 clk)
{
  u8 div;
  switch(clk)
  {
    case 16 : div = 0; break;
    case 8  : div = 1; break;
    case 4  : div = 2; break;
    case 2  : div = 3; break;
    default : div = 3; break;
  }
  CLK_CKDIVR = div;
  if(clk > 4)fac_us = (clk - 4)/4;
  else fac_us = 1;
  delay_ms(1);
}

// microseconds delay
void delay_us(u16 nus)
{
#asm
PUSH A
DELAY_XUS:
LD A,_fac_us
DELAY_US_1:
NOP
DEC A
JRNE DELAY_US_1
NOP
DECW X
JRNE DELAY_XUS
POP A
#endasm
}

// millisecond elay
void delay_ms(u16 nms)
{
  u8 t;
  if(nms>65){
    t=nms/65;
    while(t--)delay_us(65000);
    nms=nms%65;
  }
  delay_us(nms*1000);
}
```

```c
#include "ad9833.h"

void AD9833_port(void){
  //set SPI ports as push-pull outputs
  PD_DDR |= SETBIT5;  // RFSYNC (chipselect)
  PD_CR1 |= SETBIT5;

  PD_DDR |= SETBIT6;  // RSCLK
  PD_CR1 |= SETBIT6;

  PD_DDR |= SETBIT7;  // RSDATA
  PD_CR1 |= SETBIT7;

  FSYNC_SET;    // set chipselect high
  RSCLK_CLR;    // set clock low
  RSDATA_CLR;   // clear data bit
}

void AD9833_write(u16 data){
  u8 i;
  FSYNC_CLR;    // set FSYNC low
  // SENDING DATA
  for(i=0;i<16;i++){        // bit selection data
    RSCLK_SET;              // set clock high
    if(data&((u16)1<<15)){  // compare selected bit MSBs first
      RSDATA_SET;           // set data bit logic 1
    }
    else{
      RSDATA_CLR;           // set data bit logic 0
    }
    RSCLK_CLR;   // set clock LOW to accept data bit
    data<<=1;    // shift for next bit comparison
  }
  FSYNC_SET;     // set chipselect high
  RSCLK_CLR;     // set clock low
  RSDATA_CLR;    // clear data bit
}

void AD9833_init (void){

  AD9833_port();

  AD9833_write(0x2100); // Control register
  AD9833_write(0x4f16); // FreqReq 0 LSB
  AD9833_write(0x44b8); // FreqReq 0 MSB
  AD9833_write(0xC000); // PhasReq 0
  AD9833_write(0x2000); // Exit Reset
}
```

```c
#include "max2831.h"
#include "delay.h"

void MAX2831_port(void)
{
  //set SPI ports as push-pull outputs
  PG_DDR |= SETBIT4;  // SHDN
  PG_CR1 |= SETBIT4;

  PG_DDR |= SETBIT5;  // RXTX
  PG_CR1 |= SETBIT5;

  PG_DDR |= SETBIT6;  // CS (chipselect)
  PG_CR1 |= SETBIT6;

  PG_DDR |= SETBIT7;  // SCLK (clock)
  PG_CR1 |= SETBIT7;

  PC_DDR |= SETBIT5;  // DIN (data)
  PC_CR1 |= SETBIT5;  // changed to PC5

  CS_SET;     // set chipselect high
  SCLK_CLR;   // set clock low
  DIN_CLR;    // clear data bit

  SHDN_CLR;   // shutdown mode
  RXTX_CLR;
}

void MAX2831_write(u8 reg, u16 data)
{
  u8 i;
  CS_CLR;        // set CS low

  // SENDING DATA
  for(i=0;i<14;i++){        // bit selection data
    SCLK_CLR;               // set clock low
    if(data&((u16)1<<13)){  // compare selected bit MSBs first
      DIN_SET;              // set data bit logic 1
    }
    else{
      DIN_CLR;   // set data bit logic 0
    }
    SCLK_SET;    // set clock high to accept data bit
    data<<=1;    // shift for next bit comparison
  }

  // SENDING REGISTER ADRESS
  for(i=0;i<4;i++){        // bit selection reg
    SCLK_CLR;              // set clock low
    if(reg&((u8)1<<3)){    // compare selected bit MSBs first
      DIN_SET;             // set data bit logic 1
    }
    else{
      DIN_CLR;   // set data bit logic 0
    }
    SCLK_SET;    // set clock high to accept data bit
    reg<<=1;     // shift for next bit comparison
  }
  CS_SET;        // set CS high
  SCLK_CLR;      // set clock low
  DIN_CLR;       // clear data bit
}


void MAX2831_init(void)
{
```

```
    MAX2831_port(); // set up SPI ports

    MAX2831_write(0,0x0740);  // PLL mode
    MAX2831_write(1,0x119a);  // Lock-detect output select
    MAX2831_write(2,0x1003);  // recommended values
    MAX2831_write(3,0x337a);  // F/N main divider
    MAX2831_write(4,0x1ff7);  // F main divider
    MAX2831_write(5,0x00a4);  // LD enable & R divider
    MAX2831_write(6,0x0020);  // Tx/Rx calibration mode
    MAX2831_write(7,0x1022);  // Tx/Rx HP/LP corner frequencies
    MAX2831_write(8,0x3421);  // SPI, RSSI, LPF
    MAX2831_write(9,0x07b5);  // enable SPI programming
    MAX2831_write(10,0x1da4); // first/second stage PA
    MAX2831_write(11,0x007f); // LNA/VGA gain control settings
    MAX2831_write(12,0x0140); // Tx VGA gain control
    MAX2831_write(13,0x0e92); // recommended values
    MAX2831_write(14,0x013b); // ref clock output/crystal fine tune
    MAX2831_write(15,0x0545); // receiver I/Q output voltage

    SHDN_CLR;    // standby mode
    RXTX_SET;

    delay_ms(1);

    SHDN_SET;    // receive mode
    RXTX_CLR;
  }
```

```c
#include "hmc630.h"

void HMC630_port(void)
{
  //set SPI ports as push-pull outputs
  //set scan ports as pull-up inputs
  PF_DDR |= SETBIT0;  //DATA
  PF_CR1 |= SETBIT0;

  PF_DDR |= SETBIT1;  //ENABLE
  PF_CR1 |= SETBIT1;

  PF_DDR &= CLRBIT4;  //Tx_SCANOUT
  PF_CR1 |= SETBIT4;

  PF_DDR |= SETBIT5;  //CLOCK
  PF_CR1 |= SETBIT5;

  PF_DDR |= SETBIT6;  //RESET
  PF_CR1 |= SETBIT6;

  PF_DDR &= CLRBIT7;  //Rx_SCANOUT
  PF_CR1 |= SETBIT7;

  RST_SET;  // clear SPI reg with reset
  RST_CLR;  // set reset low
  EN_SET;   // set enable high
  CLK_CLR;  // set clock low
  DATA_CLR; // clear data bit
}

void HMC630_write(u8 add, u8 row, u8 data)
{
  u8 read = 0;
  u8 i;
  EN_CLR;        // set enable low

  // SENDING DATA
  for(i=0;i<8;i++){        // bit selection data
    CLK_CLR;               // set clock low
    if(data&((u8)1)){      // compare selected bit LSBs first
      DATA_SET;            // set data bit logic 1
    }
    else{
      DATA_CLR;  // set data bit logic 0
    }
    CLK_SET;     // set clock high to accept data bit
    data>>=1;    // shift for next bit comparison
  }

  // SENDING ROW ADRESS
  for(i=0;i<6;i++){        // bit selection reg
    CLK_CLR;               // set clock low
    if(row&((u8)1)){       // compare selected bit LSBs first
      DATA_SET;            // set data bit logic 1
    }
    else{
      DATA_CLR;  // set data bit logic 0
    }
    CLK_SET;     // set clock high to accept data bit
    row>>=1;     // shift for next bit comparison
  }

  // SENDING WRITE/READ BIT
  CLK_CLR;       // set clock low
  if(row>23){
    DATA_CLR;    // set R/W bit to 0 (read)
```

```
      read = 1;
    }
    else{
      DATA_SET;     // set R/W bit to 1 (write)
    }
    CLK_SET;        // set clock high to accept data bit

      // SENDING CHIP ADRESS
    for(i=0;i<3;i++){           // bit selection add
      CLK_CLR;                  // set clock low
      if(add&((u8)1)){          // compare selected bit LSBs first
        DATA_SET;               // set data bit logic 1
      }
      else{
        DATA_CLR;   // set data bit logic 0
      }
      CLK_SET;      // set clock high to accept data bit
      add>>=1;      // shift for next bit comparison
    }
    if(read){
      HMC630_read();
      read = 0;
    }
    else{
      CLK_CLR;      // set clock low
      DATA_CLR;     // clear data bit
      EN_SET;       // set enable high
    }
  }


  u8 HMC630_read(void)
  {
    // For monitoring lock detect, VCO amplitude and temperature
    // not used for now

    CLK_CLR;        // set clock low
    EN_SET;         // set enable high
    DATA_CLR;       // clear data bit
  }

  void HMC630_init(void)
  {
    HMC630_port();

    // initialize hmc6300
    HMC630_write(6, 0x01, 0xca);
    HMC630_write(6, 0x02, 0xff);
    HMC630_write(6, 0x03, 0xf6);
    HMC630_write(6, 0x04, 0x00);
    HMC630_write(6, 0x05, 0xff);
    HMC630_write(6, 0x06, 0xec);
    HMC630_write(6, 0x07, 0x0f);
    HMC630_write(6, 0x08, 0x8f);
    HMC630_write(6, 0x09, 0x00);
    HMC630_write(6, 0x0a, 0x51);
    HMC630_write(6, 0x0b, 0x03);
    HMC630_write(6, 0x0c, 0x64);
    HMC630_write(6, 0x10, 0x36);
    HMC630_write(6, 0x11, 0xbb);
    HMC630_write(6, 0x12, 0x46);
    HMC630_write(6, 0x13, 0x02);
    HMC630_write(6, 0x14, 0x35);
    HMC630_write(6, 0x15, 0x12);
    HMC630_write(6, 0x16, 0x0a);
    HMC630_write(6, 0x17, 0x62);

```

```
135       // initialize hmc6301
136       HMC630_write(7, 0x00, 0x20);
137       HMC630_write(7, 0x01, 0x4c);
138       HMC630_write(7, 0x02, 0x03);
139       HMC630_write(7, 0x03, 0x03);
140       HMC630_write(7, 0x04, 0x9F);
141       HMC630_write(7, 0x05, 0xff);
142       HMC630_write(7, 0x06, 0xbf);
143       HMC630_write(7, 0x07, 0x6d);
144       HMC630_write(7, 0x08, 0x80);
145       HMC630_write(7, 0x09, 0x40);
146       HMC630_write(7, 0x10, 0x36);
147       HMC630_write(7, 0x11, 0xbb);
148       HMC630_write(7, 0x12, 0x46);
149       HMC630_write(7, 0x13, 0x02);
150       HMC630_write(7, 0x14, 0x2b);
151       HMC630_write(7, 0x15, 0x12);
152       HMC630_write(7, 0x16, 0x05);
153       HMC630_write(7, 0x17, 0x62);
154     }
155
```

```c
#include "HC05.h"
#include "delay.h"

// initalize the UART ports used for USB
// enables transmitter & receiver
void UART_Init(void)
{
    //UART Baudrate definition (done here is faster)
    #define BAUDRATE 115200 // can be changed to higher
    #define F_CLK 16000000
    enum
    {
      BAUDRATE_HSI1_BRR2 = ((F_CLK/BAUDRATE)/4096)|((F_CLK/BAUDRATE)%16),
      BAUDRATE_HSI1_BRR1 = (F_CLK/BAUDRATE)/16
    };

    PC_DDR |= SETBIT2; //Output RX
    PC_CR1 &= CLRBIT2; //open drain
    PC_CR2 |= SETBIT2; //up to 10MHZ

    PC_CR2 &= CLRBIT3; //external interrupt disabled
    PC_DDR &= CLRBIT3; //Input TX
    PC_CR1 &= CLRBIT3; //floating

    SYSCFG_RMPCR1 &= CLRBIT5; // USART1_TX on PC3
    SYSCFG_RMPCR1 &= CLRBIT4; // USART1_RX on PC2

    CLK_PCKENR1 |= SETBIT5; // enable system clock on uart1

    USART1_CR2 &= CLRBIT2; // Receiver is disabled
    USART1_CR2 &= CLRBIT3; // Transmitter is disabled

    //configure baudrate HSI-16M
    USART1_BRR2 = BAUDRATE_HSI1_BRR2;
    USART1_BRR1 = BAUDRATE_HSI1_BRR1;

    USART1_CR1 &= CLRBIT4; // 8 data bits word length
    USART1_CR3 &= CLRBIT4; // 1 STOP bit
    USART1_CR3 &= CLRBIT5;
    USART1_CR1 &= CLRBIT2; // Parity control disabled
    USART1_CR2 &= CLRBIT6; //disable transmission interruption
    USART1_CR2 |= SETBIT5; //enable receiver interruption

    USART1_CR1 &= CLRBIT5; // USART enabled

    USART1_CR2 |= SETBIT2; //Receiver is enabled and begins searching for a start bit
    USART1_CR2 |= SETBIT3; //Transmitter is enabled

    //interrupt priority
    ITC_SPR7 &= CLRBIT7; // VECT27SPR
    ITC_SPR7 |= SETBIT6;
    ITC_SPR8 &= CLRBIT1; // VECT28SPR
    ITC_SPR8 |= SETBIT0;

    USART1_SR &= CLRBIT3; // No Overrun error
    USART1_SR &= CLRBIT5; // Data is not received
}


// send string through UART stop at length chars
void UART_SendStringLenght(char* Data,u8 length)
{
  u8 i = 0;
  //send string byte per byte for length char
  for(i = 0; i<length;i++){
    //wait till last byte is send
    while((USART1_SR&CHSBIT7)==0x00);
```

```
68          //put the next byte into usart register
69          USART1_DR = Data[i];
70        }
71      }
72
```

```c
#include "adc.h"

//Init the adc peripheral
void ADC_Init(void)
{
    CLK_PCKENR2 |= SETBIT0; //enable system clock on ADC1

    ADC1_SR = 0x00; //clear status register

    //12 bits, no interrupts, single conversion mode
    ADC1_CR1 = 0b00000000;

    //no tirgger, sampling time selection in clock cycles
    ADC1_CR2 = 0b00000000 | ADC_CLOCK_48; //f(adc) = CK

    //no channel 24, default channel selection
    ADC1_CR3 = 0b00000000;

    //set all adc pin as pull up input
    PA_DDR &= 0b10001111; //PA4-6
    PA_CR1 |= 0b01110000; //pull up input

    PC_DDR &= 0b11110110; //PC4 & PC7
    PC_CR1 |= 0b00001001; //pull up input

}

// define channel to convert
void ADC_On_SetChannel(u8 Channel)
{
    //Enable ADC by setting the ADON bit
    ADC1_CR1 |= SETBIT0;

    //clear the channel selection bit
    ADC1_SQR1 = 0x00;
    ADC1_SQR2 = 0x00;
    ADC1_SQR3 = 0x00;
    ADC1_SQR4 = 0x00;

    //select the correct channel
    if (Channel>23){
      ADC1_SQR1 |= (1<<(Channel-24));
    }
    else if (Channel>15){
      ADC1_SQR2 |= (1<<(Channel-16));
    }
    else if (Channel>7){
      ADC1_SQR3|= (1<<(Channel-8));
    }
    else {
      ADC1_SQR4 |= (1<<(Channel));
    }
}

// reading last converted data, return 1 when ready
bool ADC_Get(u16 *Value)
{
    ADC1_CR1 |= SETBIT1;     // start the conversion

    while(!(ADC1_SR&0b1));  // wait until convertion is done

    ADC1_SR &= CLRBIT0;      // reset EOC flag

    // put 12 bits converted data on Value
    *Value = ((ADC1_DRH&0b00001111)<<8)+ADC1_DRL;

    ADC1_CR1 &= CLRBIT0;    //disable ADC
```

```
        return 1;
    }
```

```
#ifndef  __DELAY_H
#define  __DELAY_H
#include "prohead.h"

// initialize clock
void CLK_Init(u8 clk);

// microseconds delay
void delay_us(u16 nus);

// millisecond elay
void delay_ms(u16 nms);
#endif

```

```
1   #ifndef __AD9833_H
2   #define __AD9833_H
3   #include "prohead.h"
4
5   //PD5 -> FSYNC  chipselect
6   //PD6 -> RSCLK  clock
7   //PD7 -> RSDATA data
8
9   #define FSYNC_SET     PD_ODR |= SETBIT5;
10  #define FSYNC_CLR     PD_ODR &= CLRBIT5;
11  #define RSCLK_SET     PD_ODR |= SETBIT6;
12  #define RSCLK_CLR     PD_ODR &= CLRBIT6;
13  #define RSDATA_SET    PD_ODR |= SETBIT7;
14  #define RSDATA_CLR    PD_ODR &= CLRBIT7;
15
16  // Inisialize the serial ports
17  // function used in init
18  void AD9833_port(void);
19
20  // write command line
21  // function used in init
22  void AD9833_write(u16 data);
23
24  // load all the data through SPI into AD9833
25  void AD9833_init (void);
26
27
28  #endif
```

```c
#ifndef __MAX2831_H
#define __MAX2831_H
#include "prohead.h"

//PG4 -> SHDN   modeselect
//PG5 -> RXTX   modeselect
//PG6 -> CS     chipselect
//PG7 -> SCLK   clock
//PC5 -> DIN    data

#define SHDN_SET    PG_ODR |= SETBIT4;
#define SHDN_CLR    PG_ODR &= CLRBIT4;
#define RXTX_SET    PG_ODR |= SETBIT5;
#define RXTX_CLR    PG_ODR &= CLRBIT5;
#define CS_SET      PG_ODR |= SETBIT6;
#define CS_CLR      PG_ODR &= CLRBIT6;
#define SCLK_SET    PG_ODR |= SETBIT7;
#define SCLK_CLR    PG_ODR &= CLRBIT7;
#define DIN_SET     PC_ODR |= SETBIT5;
#define DIN_CLR     PC_ODR &= CLRBIT5;

// Inisialize the serial ports
// function used in init
void MAX2831_port(void);

// write command line used
// function used in init
void MAX2831_write(u8 reg, u16 data);

// load all the data through SPI into MAX2831
void MAX2831_init(void);

#endif
```

```
1   #ifndef __HMC630_H
2   #define __HMC630_H
3   #include "prohead.h"
4
5   //PF0 -> DATA
6   //PF1 -> ENABBLE
7   //PF4 <- Tx SCANOUT
8   //PF5 -> CLOCK
9   //PF6 -> RESET
10  //PF7 <- Rx_SCANOUT
11
12  #define DATA_SET    PF_ODR |= SETBIT0;
13  #define DATA_CLR    PF_ODR &= CLRBIT0;
14  #define EN_SET      PF_ODR |= SETBIT1;
15  #define EN_CLR      PF_ODR &= CLRBIT1;
16  #define TSCAN_SET   PF_ODR |= SETBIT4;
17  #define TSCAN_CLR   PF_ODR &= CLRBIT4;
18  #define CLK_SET     PF_ODR |= SETBIT5;
19  #define CLK_CLR     PF_ODR &= CLRBIT5;
20  #define RST_SET     PF_ODR |= SETBIT6;
21  #define RST_CLR     PF_ODR &= CLRBIT6;
22  #define RSCAN_SET   PF_ODR |= SETBIT7;
23  #define RSCAN_CLR   PF_ODR &= CLRBIT7;
24
25  // Inisialize the serial ports
26  // function used in init
27  void HMC630_port(void);
28
29  // write command line
30  // function used in init
31  void HMC630_write(u8 row, u8 reg, u8 data);
32
33  // read function
34  u8 HMC630_read(void);
35
36  // load all the data through SPI into HMC6300 and HMC6301
37  void HMC630_init(void);
38
39
40  #endif
```

```
/*****************************************************************************/
/**
 * \file HC05.h
 * \brief provide UART parameters and communication with HC05 bluetooth-uart
 * \version 1.0
 * \date 21.07.2017
 * \author Lucas Bonvin
 */
/*****************************************************************************/

#ifndef  __HC05_H
#define  __HC05_H
#include "prohead.h"


//UART Baudrate definition
#define BR9600 9600
#define BR38400 38400
#define BR57600 57600
#define BR115200 115200
#define F_CLK 16000000
enum
{
    BAUDRATE_HSI1_BRR2_9600 = ((F_CLK/BR9600)/4096)|((F_CLK/BR9600)%16),
    BAUDRATE_HSI1_BRR1_9600 = (F_CLK/BR9600)/16,
    BAUDRATE_HSI1_BRR2_38400 = ((F_CLK/BR38400)/4096)|((F_CLK/BR38400)%16),
    BAUDRATE_HSI1_BRR1_38400 = (F_CLK/BR38400)/16,
    BAUDRATE_HSI1_BRR2_57600 = ((F_CLK/BR57600)/4096)|((F_CLK/BR57600)%16),
    BAUDRATE_HSI1_BRR1_57600 = (F_CLK/BR57600)/16,
    BAUDRATE_HSI1_BRR2_115200 = ((F_CLK/BR115200)/4096)|((F_CLK/BR115200)%16),
    BAUDRATE_HSI1_BRR1_115200 = (F_CLK/BR115200)/16
};


// initalize the UART ports used for USB
// enables transmitter & receiver
void UART_Init(void);

// send string through UART stop at length chars
void UART_SendStringLenght(char* Data,u8 length);


#endif
```

```c
#ifndef  __ADC_H
#define  __ADC_H
#include "prohead.h"

//Initialize the adc peripheral
void ADC_Init(void);

// define channel sequence and START conversion
void ADC_On_SetChannel(u8 Channel);

// reading last converted data value
bool ADC_Get(u16 *Value);

#endif
```

```
/*****************************************************************************/
/**
 * \file prohead.h
 * \brief Provide various definition of the system
 * \version 1.0
 * \date 07.07.2017
 * \author Lucas Bonvin
 * \author ZangMei
 * \author Marcel Balle
 */
/*****************************************************************************/

#ifndef __ProHead__
#define __ProHead__

#include "STM8L151R8.h"
//++++++++++++++++++++++++++ Standard C LIBS +++++++++++++++++++++++++++++++++/
#include <string.h>
#include <stdio.h>
#include <stddef.h>

//++++++++++++++++++++++++++ Numbers Define  +++++++++++++++++++++++++++++++++//
#define  True         1
#define  False        0
#define  ZERO         0
#define  ONE          1
#define  TWO          2
#define  THREE        3
#define  FOUR         4
#define  FIVE         5
#define  SIX          6
#define  SEVEN        7
#define  EIGHT        8
#define  NINE         9
#define  TEN          10
#define  HUNDRED      100
#define  THOUSAND     1000
#define  HALFBYTEMAX  15
#define  ONEBYTEMAX   255

//++++++++++++++++++++++++++ Type Declaration +++++++++++++++++++++++++++++++++//
//!You should modify it for different c compiler.
typedef unsigned char       bool;
typedef          char       ascii;
typedef unsigned char       u8;
typedef signed   char       s8;
typedef unsigned short      u16;
typedef signed   short      s16;
typedef unsigned long       u32;
typedef signed   long       s32;

//++++++++++++++++++++++++++ Bit Computing +++++++++++++++++++++++++++++++++//
//SET BIT.    Example: a |= SETBIT0
enum
{
  SETBIT0 = 0x0001,  SETBIT1 = 0x0002,  SETBIT2 = 0x0004,  SETBIT3 = 0x0008,
  SETBIT4 = 0x0010,  SETBIT5 = 0x0020,  SETBIT6 = 0x0040,  SETBIT7 = 0x0080,
  SETBIT8 = 0x0100,  SETBIT9 = 0x0200,  SETBIT10 = 0x0400, SETBIT11 = 0x0800,
  SETBIT12 = 0x1000, SETBIT13 = 0x2000, SETBIT14 = 0x4000, SETBIT15 = 0x8000
};
//CLR BIT.    Example: a &= CLRBIT0
enum
{
  CLRBIT0 = 0xFFFE,  CLRBIT1 = 0xFFFD,  CLRBIT2 = 0xFFFB,  CLRBIT3 = 0xFFF7,
  CLRBIT4 = 0xFFEF,  CLRBIT5 = 0xFFDF,  CLRBIT6 = 0xFFBF,  CLRBIT7 = 0xFF7F,
  CLRBIT8 = 0xFEFF,  CLRBIT9 = 0xFDFF,  CLRBIT10 = 0xFBFF, CLRBIT11 = 0xF7FF,
  CLRBIT12 = 0xEFFF, CLRBIT13 = 0xDFFF, CLRBIT14 = 0xBFFF, CLRBIT15 = 0x7FFF
```

```
};
//CHOSE BIT.  Example: a = b&CHSBIT0
enum
{
  CHSBIT0 = 0x0001,  CHSBIT1 = 0x0002,  CHSBIT2 = 0x0004,  CHSBIT3 = 0x0008,
  CHSBIT4 = 0x0010,  CHSBIT5 = 0x0020,  CHSBIT6 = 0x0040,  CHSBIT7 = 0x0080,
  CHSBIT8 = 0x0100,  CHSBIT9 = 0x0200,  CHSBIT10 = 0x0400, CHSBIT11 = 0x0800,
  CHSBIT12 = 0x1000, CHSBIT13 = 0x2000, CHSBIT14 = 0x4000, CHSBIT15 = 0x8000
};

//+++++++++++++++++++++++++++++ Others +++++++++++++++++++++++++++++++++++//
//TAST RUN STEPS.
enum
{
  STEP0 = 0,          STEP1 = 1,          STEP2 = 2,          STEP3 = 3,
  STEP4 = 4,          STEP5 = 5,          STEP6 = 6,          STEP7 = 7,
  STEP8 = 8,          STEP9 = 9,          STEP10 = 10,        STEP11 = 11,
  STEP12 = 12,        STEP13 = 13,        STEP14 = 14,        STEP15 = 15,
  STEP16 = 16,        STEP17 = 17,        STEP18 = 18,        STEP19 = 19,
  STEP20 = 20,        STEP21 = 21,        STEP22 = 22,        STEP23 = 23,
  STEP24 = 24,        STEP25 = 25,        STEP26 = 26,        STEP27 = 27,
  STEP28 = 28,        STEP29 = 29,        STEP30 = 30,        STEP31 = 31,
  STEP32 = 32,        STEP33 = 33,        STEP34 = 34,        STEP35 = 35
};

//+++++++++++++++++++++++++++++ ADC +++++++++++++++++++++++++++++++++++//
//Sampling time selection
enum
{
  ADC_CLOCK_4  = 0,         ADC_CLOCK_9  = 1,         ADC_CLOCK_16  = 2,         ADC_CLOCK_24  = 3,
  ADC_CLOCK_48 = 4,         ADC_CLOCK_96 = 5,         ADC_CLOCK_192 = 6,         ADC_CLOCK_384 = 7
};

//+++++++++++++++++++++++++++++ ADC +++++++++++++++++++++++++++++++++++//
//Aampling time selection
enum
{
  FSYSCLK_DIVIDE_2  =  0,   FSYSCLK_DIVIDE_4  =  8,   FSYSCLK_DIVIDE_8   = 16,   FSYSCLK_DIVIDE_16  = 24,
  FSYSCLK_DIVIDE_32 = 32,   FSYSCLK_DIVIDE_64 = 40,   FSYSCLK_DIVIDE_128 = 48,   FSYSCLK_DIVIDE_256 = 56
};

//++++++++++++++++++++++++++++System definition++++++++++++++++++++++++++//
// Physical value
#define CLK_F 16          //Change to actual clk frequency
#define U_ALIM 33000      //Change to actual alim voltage
#define U_MAX_INPUT 35000

//EEprom buffer
#define N_DATA_TO_SAVE 10

//ADC Definition
#define ADC_RESOLUTION 4096   // 12 bits resolution
#define N_ADC_CHANNEL 5       //Change to actual channel number

#define MAX_ITERATION_OPTI 255

#endif
```

```
/* STM8L151.h */

/* Copyright (c) 2003-2012 STMicroelectronics */

#ifndef __STM8L151x8__
#define __STM8L151x8__

/* STM8L151 */

  /* Check MCU name */
  #ifdef MCU_NAME
    #define STM8L151 1
    #if (MCU_NAME != STM8L151)
    #error "wrong include file STM8L151x8.h for chosen MCU!"
    #endif
  #endif

  #ifdef __CSMC__
    #define DEF_8BIT_REG_AT(NAME,ADDRESS) volatile unsigned char NAME @ADDRESS
    #define DEF_16BIT_REG_AT(NAME,ADDRESS) volatile unsigned int NAME @ADDRESS
  #endif

  #ifdef __RAISONANCE__
    #define DEF_8BIT_REG_AT(NAME,ADDRESS) at ADDRESS hreg NAME
    #define DEF_16BIT_REG_AT(NAME,ADDRESS) at ADDRESS hreg16 NAME
  #endif

/* Port A */
/*****************************************************************/

/* Port A data output latch register */
DEF_8BIT_REG_AT(PA_ODR,0x5000);

/* Port A input pin value register */
DEF_8BIT_REG_AT(PA_IDR,0x5001);

/* Port A data direction register */
DEF_8BIT_REG_AT(PA_DDR,0x5002);

/* Port A control register 1 */
DEF_8BIT_REG_AT(PA_CR1,0x5003);

/* Port A control register 2 */
DEF_8BIT_REG_AT(PA_CR2,0x5004);

/* Port B */
/*****************************************************************/

/* Port B data output latch register */
DEF_8BIT_REG_AT(PB_ODR,0x5005);

/* Port B input pin value register */
DEF_8BIT_REG_AT(PB_IDR,0x5006);

/* Port B data direction register */
DEF_8BIT_REG_AT(PB_DDR,0x5007);

/* Port B control register 1 */
DEF_8BIT_REG_AT(PB_CR1,0x5008);

/* Port B control register 2 */
DEF_8BIT_REG_AT(PB_CR2,0x5009);

/* Port C */
/*****************************************************************/

/* Port C data output latch register */
```

```
    DEF_8BIT_REG_AT(PC_ODR,0x500a);

    /* Port C input pin value register */
    DEF_8BIT_REG_AT(PC_IDR,0x500b);

    /* Port C data direction register */
    DEF_8BIT_REG_AT(PC_DDR,0x500c);

    /* Port C control register 1 */
    DEF_8BIT_REG_AT(PC_CR1,0x500d);

    /* Port C control register 2 */
    DEF_8BIT_REG_AT(PC_CR2,0x500e);

    /* Port D */
    /*****************************************************************/

    /* Port D data output latch register */
    DEF_8BIT_REG_AT(PD_ODR,0x500f);

    /* Port D input pin value register */
    DEF_8BIT_REG_AT(PD_IDR,0x5010);

    /* Port D data direction register */
    DEF_8BIT_REG_AT(PD_DDR,0x5011);

    /* Port D control register 1 */
    DEF_8BIT_REG_AT(PD_CR1,0x5012);

    /* Port D control register 2 */
    DEF_8BIT_REG_AT(PD_CR2,0x5013);

    /* Port E */
    /*****************************************************************/

    /* Port E data output latch register */
    DEF_8BIT_REG_AT(PE_ODR,0x5014);

    /* Port E input pin value register */
    DEF_8BIT_REG_AT(PE_IDR,0x5015);

    /* Port E data direction register */
    DEF_8BIT_REG_AT(PE_DDR,0x5016);

    /* Port E control register 1 */
    DEF_8BIT_REG_AT(PE_CR1,0x5017);

    /* Port E control register 2 */
    DEF_8BIT_REG_AT(PE_CR2,0x5018);

    /* Port F */
    /*****************************************************************/

    /* Port F data output latch register */
    DEF_8BIT_REG_AT(PF_ODR,0x5019);

    /* Port F input pin value register */
    DEF_8BIT_REG_AT(PF_IDR,0x501a);

    /* Port F data direction register */
    DEF_8BIT_REG_AT(PF_DDR,0x501b);

    /* Port F control register 1 */
    DEF_8BIT_REG_AT(PF_CR1,0x501c);

    /* Port F control register 2 */
    DEF_8BIT_REG_AT(PF_CR2,0x501d);
```

```
135
136     /* Port G */
137     /*****************************************************************/
138
139     /* Port G data output latch register */
140     DEF_8BIT_REG_AT(PG_ODR,0x501e);
141
142     /* Port G input pin value register */
143     DEF_8BIT_REG_AT(PG_IDR,0x501f);
144
145     /* Port G data direction register */
146     DEF_8BIT_REG_AT(PG_DDR,0x5020);
147
148     /* Port G control register 1 */
149     DEF_8BIT_REG_AT(PG_CR1,0x5021);
150
151     /* Port G control register 2 */
152     DEF_8BIT_REG_AT(PG_CR2,0x5022);
153
154     /* Port H */
155     /*****************************************************************/
156
157     /* Port H data output latch register */
158     DEF_8BIT_REG_AT(PH_ODR,0x5023);
159
160     /* Port H input pin value register */
161     DEF_8BIT_REG_AT(PH_IDR,0x5024);
162
163     /* Port H data direction register */
164     DEF_8BIT_REG_AT(PH_DDR,0x5025);
165
166     /* Port H control register 1 */
167     DEF_8BIT_REG_AT(PH_CR1,0x5026);
168
169     /* Port H control register 2 */
170     DEF_8BIT_REG_AT(PH_CR2,0x5027);
171
172     /* Port I */
173     /*****************************************************************/
174
175     /* Port I data output latch register */
176     DEF_8BIT_REG_AT(PI_ODR,0x5028);
177
178     /* Port I input pin value register */
179     DEF_8BIT_REG_AT(PI_IDR,0x5029);
180
181     /* Port I data direction register */
182     DEF_8BIT_REG_AT(PI_DDR,0x502a);
183
184     /* Port I control register 1 */
185     DEF_8BIT_REG_AT(PI_CR1,0x502b);
186
187     /* Port I control register 2 */
188     DEF_8BIT_REG_AT(PI_CR2,0x502c);
189
190     /* Flash */
191     /*****************************************************************/
192
193     /* Flash control register 1 */
194     DEF_8BIT_REG_AT(FLASH_CR1,0x5050);
195
196     /* Flash control register 2 */
197     DEF_8BIT_REG_AT(FLASH_CR2,0x5051);
198
199     /* Flash Program memory unprotection register */
200     DEF_8BIT_REG_AT(FLASH_PUKR,0x5052);
201
```

```
    /* Data EEPROM unprotection register */
    DEF_8BIT_REG_AT(FLASH_DUKR,0x5053);

    /* Flash in-application programming status register */
    DEF_8BIT_REG_AT(FLASH_IAPSR,0x5054);

    /* Direct memory access controller 1 (DMA1) */
    /*****************************************************************/

    /* DMA1 global configuration & status register */
    DEF_8BIT_REG_AT(DMA1_GCSR,0x5070);

    /* DMA1 global interrupt register 1 */
    DEF_8BIT_REG_AT(DMA1_GIR1,0x5071);

    /* DMA1 channel 0 configuration register */
    DEF_8BIT_REG_AT(DMA1_C0CR,0x5075);

    /* DMA1 channel 0 status & priority register */
    DEF_8BIT_REG_AT(DMA1_C0SPR,0x5076);

    /* DMA1 number of data to transfer register (channel 0) */
    DEF_8BIT_REG_AT(DMA1_C0NDTR,0x5077);

    /* DMA1 peripheral address register (channel 0) */
    DEF_16BIT_REG_AT(DMA1_C0PAR,0x5078);
    /* DMA peripheral address high register (channel 0) */
    DEF_8BIT_REG_AT(DMA1_C0PARH,0x5078);
    /* DMA peripheral address low register (channel 0) */
    DEF_8BIT_REG_AT(DMA1_C0PARL,0x5079);

    /* DMA1 memory 0 address register (channel 0) */
    DEF_16BIT_REG_AT(DMA1_C0M0AR,0x507b);
    /* DMA memory address high register (channel 0) */
    DEF_8BIT_REG_AT(DMA1_C0M0ARH,0x507b);
    /* DMA memory address low register (channel 0) */
    DEF_8BIT_REG_AT(DMA1_C0M0ARL,0x507c);

    /* DMA1 channel 1 configuration register */
    DEF_8BIT_REG_AT(DMA1_C1CR,0x507f);

    /* DMA1 channel 1 status & priority register */
    DEF_8BIT_REG_AT(DMA1_C1SPR,0x5080);

    /* DMA1 number of data to transfer register (channel 1) */
    DEF_8BIT_REG_AT(DMA1_C1NDTR,0x5081);

    /* DMA1 peripheral address register (channel 1) */
    DEF_16BIT_REG_AT(DMA1_C1PAR,0x5082);
    /* DMA peripheral address high register (channel 1) */
    DEF_8BIT_REG_AT(DMA1_C1PARH,0x5082);
    /* DMA peripheral address low register (channel 1) */
    DEF_8BIT_REG_AT(DMA1_C1PARL,0x5083);

    /* DMA1 memory 0 address register (channel 1) */
    DEF_16BIT_REG_AT(DMA1_C1M0AR,0x5085);
    /* DMA memory address high register (channel 1) */
    DEF_8BIT_REG_AT(DMA1_C1M0ARH,0x5085);
    /* DMA memory address low register (channel 1) */
    DEF_8BIT_REG_AT(DMA1_C1M0ARL,0x5086);

    /* DMA1 channel 2 configuration register */
    DEF_8BIT_REG_AT(DMA1_C2CR,0x5089);

    /* DMA1 channel 2 status & priority register */
    DEF_8BIT_REG_AT(DMA1_C2SPR,0x508a);

```

```
    /* DMA1 number of data to transfer register (channel 2) */
    DEF_8BIT_REG_AT(DMA1_C2NDTR,0x508b);

    /* DMA1 peripheral address register (channel 2) */
    DEF_16BIT_REG_AT(DMA1_C2PAR,0x508c);
    /* DMA peripheral address high register (channel 2) */
    DEF_8BIT_REG_AT(DMA1_C2PARH,0x508c);
    /* DMA peripheral address low register (channel 2) */
    DEF_8BIT_REG_AT(DMA1_C2PARL,0x508d);

    /* DMA1 memory 0 address register (channel 2) */
    DEF_16BIT_REG_AT(DMA1_C2M0AR,0x508f);
    /* DMA memory address high register (channel 2) */
    DEF_8BIT_REG_AT(DMA1_C2M0ARH,0x508f);
    /* DMA memory address low register (channel 2) */
    DEF_8BIT_REG_AT(DMA1_C2M0ARL,0x5090);

    /* DMA1 channel 3 configuration register */
    DEF_8BIT_REG_AT(DMA1_C3CR,0x5093);

    /* DMA1 channel 3 status & priority register */
    DEF_8BIT_REG_AT(DMA1_C3SPR,0x5094);

    /* DMA1 number of data to transfer register (channel 3) */
    DEF_8BIT_REG_AT(DMA1_C3NDTR,0x5095);

    /* DMA1 peripheral address register (channel 3) */
    DEF_16BIT_REG_AT(DMA1_C3PAR_C3M1AR,0x5096);
    /* DMA1 peripheral address high register (channel 3) */
    DEF_8BIT_REG_AT(DMA1_C3PARH_C3M1ARH,0x5096);
    /* DMA1 peripheral address low register (channel 3) */
    DEF_8BIT_REG_AT(DMA1_C3PARL_C3M1ARL,0x5097);

    /* DMA1 memory 0 address register (channel 3) */
    DEF_16BIT_REG_AT(DMA1_C3M0AR,0x5099);
    /* DMA memory address high register (channel 3) */
    DEF_8BIT_REG_AT(DMA1_C3M0ARH,0x5099);
    /* DMA memory address low register (channel 3) */
    DEF_8BIT_REG_AT(DMA1_C3M0ARL,0x509a);

    /* System configuration (SYSCFG) */
    /*****************************************************************/

    /* Remapping register 3 */
    DEF_8BIT_REG_AT(SYSCFG_RMPCR3,0x509d);

    /* Remapping register 1 */
    DEF_8BIT_REG_AT(SYSCFG_RMPCR1,0x509e);

    /* Remapping register 2 */
    DEF_8BIT_REG_AT(SYSCFG_RMPCR2,0x509f);

    /* External Interrupt Control Register (ITC) */
    /*****************************************************************/

    /* External interrupt control register 1 */
    DEF_8BIT_REG_AT(EXTI_CR1,0x50a0);

    /* External interrupt control register 2 */
    DEF_8BIT_REG_AT(EXTI_CR2,0x50a1);

    /* External interrupt control register 3 */
    DEF_8BIT_REG_AT(EXTI_CR3,0x50a2);

    /* External interrupt status register 1 */
    DEF_8BIT_REG_AT(EXTI_SR1,0x50a3);

```

```
    /* External interrupt status register 2 */
    DEF_8BIT_REG_AT(EXTI_SR2,0x50a4);

    /* External interrupt port select register 1 */
    DEF_8BIT_REG_AT(EXTI_CONF1,0x50a5);

    /* Wait For Event (WFE) */
    /*****************************************************************/

    /* WFE control register 1 */
    DEF_8BIT_REG_AT(WFE_CR1,0x50a6);

    /* WFE control register 2 */
    DEF_8BIT_REG_AT(WFE_CR2,0x50a7);

    /* WFE control register 3 */
    DEF_8BIT_REG_AT(WFE_CR3,0x50a8);

    /* WFE control register 4 */
    DEF_8BIT_REG_AT(WFE_CR4,0x50a9);

    /*  External interrupt (ITC-EXTI) */
    /*****************************************************************/

    /* External interrupt control register 4 */
    DEF_8BIT_REG_AT(EXTI_CR4,0x50aa);

    /* External interrupt port select register 2 */
    DEF_8BIT_REG_AT(EXTI_CONF2,0x50ab);

    /* Reset (RST) */
    /*****************************************************************/

    /* Reset control register */
    DEF_8BIT_REG_AT(RST_CR,0x50b0);

    /* Reset status register */
    DEF_8BIT_REG_AT(RST_SR,0x50b1);

    /* Power control (PWR) */
    /*****************************************************************/

    /* Power control and status register 1 */
    DEF_8BIT_REG_AT(PWR_CSR1,0x50b2);

    /* Power control and status register 2 */
    DEF_8BIT_REG_AT(PWR_CSR2,0x50b3);

    /* Clock Control (CLK) */
    /*****************************************************************/

    /* System clock divider register */
    DEF_8BIT_REG_AT(CLK_CKDIVR,0x50c0);

    /* Clock RTC register */
    DEF_8BIT_REG_AT(CLK_CRTCR,0x50c1);

    /* Internal clock control register */
    DEF_8BIT_REG_AT(CLK_ICKCR,0x50c2);

    /* Peripheral clock gating register 1 */
    DEF_8BIT_REG_AT(CLK_PCKENR1,0x50c3);

    /* Peripheral clock gating register 2 */
    DEF_8BIT_REG_AT(CLK_PCKENR2,0x50c4);

    /* Configurable clock control register */
```

```
    DEF_8BIT_REG_AT(CLK_CCOR,0x50c5);

    /* External clock control register */
    DEF_8BIT_REG_AT(CLK_ECKCR,0x50c6);

    /* System clock status register */
    DEF_8BIT_REG_AT(CLK_SCSR,0x50c7);

    /* System clock switch register */
    DEF_8BIT_REG_AT(CLK_SWR,0x50c8);

    /* Clock switch control register */
    DEF_8BIT_REG_AT(CLK_SWCR,0x50c9);

    /* Clock security system register */
    DEF_8BIT_REG_AT(CLK_CSSR,0x50ca);

    /* Clock BEEP register */
    DEF_8BIT_REG_AT(CLK_CBEEPR,0x50cb);

    /* HSI calibration register */
    DEF_8BIT_REG_AT(CLK_HSICALR,0x50cc);

    /* HSI clock calibration trimming register */
    DEF_8BIT_REG_AT(CLK_HSITRIMR,0x50cd);

    /* HSI unlock register */
    DEF_8BIT_REG_AT(CLK_HSIUNLCKR,0x50ce);

    /* Main regulator control status register */
    DEF_8BIT_REG_AT(CLK_REGCSR,0x50cf);

    /* Window Watchdog (WWDG) */
    /*****************************************************************/

    /* WWDG Control Register */
    DEF_8BIT_REG_AT(WWDG_CR,0x50d3);

    /* WWDR Window Register */
    DEF_8BIT_REG_AT(WWDG_WR,0x50d4);

    /* Independent Watchdog (IWDG) */
    /*****************************************************************/

    /* IWDG Key Register */
    DEF_8BIT_REG_AT(IWDG_KR,0x50e0);

    /* IWDG Prescaler Register */
    DEF_8BIT_REG_AT(IWDG_PR,0x50e1);

    /* IWDG Reload Register */
    DEF_8BIT_REG_AT(IWDG_RLR,0x50e2);

    /* Beeper (BEEP) */
    /*****************************************************************/

    /* BEEP Control/Status Register 1 */
    DEF_8BIT_REG_AT(BEEP_CSR1,0x50f0);

    /* BEEP Control/Status Register 2 */
    DEF_8BIT_REG_AT(BEEP_CSR2,0x50f3);

    /* Real-time clock (RTC) */
    /*****************************************************************/

    /* Time Register 1 */
    DEF_8BIT_REG_AT(RTC_TR1,0x5140);
```

```
470
471     /* Time Register 2 */
472     DEF_8BIT_REG_AT(RTC_TR2,0x5141);
473
474     /* Time Register 3 */
475     DEF_8BIT_REG_AT(RTC_TR3,0x5142);
476
477     /* Date Register 1 */
478     DEF_8BIT_REG_AT(RTC_DR1,0x5144);
479
480     /* Date Register 2 */
481     DEF_8BIT_REG_AT(RTC_DR2,0x5145);
482
483     /* Date Register 3 */
484     DEF_8BIT_REG_AT(RTC_DR3,0x5146);
485
486     /* Control Register 1 */
487     DEF_8BIT_REG_AT(RTC_CR1,0x5148);
488
489     /* Control Register 2 */
490     DEF_8BIT_REG_AT(RTC_CR2,0x5149);
491
492     /* Control Register 3 */
493     DEF_8BIT_REG_AT(RTC_CR3,0x514a);
494
495     /* Initialization and Status Register 1 */
496     DEF_8BIT_REG_AT(RTC_ISR1,0x514c);
497
498     /* Initialization and Status Register 2 */
499     DEF_8BIT_REG_AT(RTC_ISR2,0x514d);
500
501     /* Synchronous Prescaler Register */
502     DEF_16BIT_REG_AT(RTC_SPRER,0x5150);
503     /* Synchronous Prescaler Register High */
504     DEF_8BIT_REG_AT(RTC_SPRERH,0x5150);
505     /* Synchronous Prescaler Register Low */
506     DEF_8BIT_REG_AT(RTC_SPRERL,0x5151);
507
508     /* Asynchronous Prescaler Register */
509     DEF_8BIT_REG_AT(RTC_APRER,0x5152);
510
511     /* Wakeup Timer Register */
512     DEF_16BIT_REG_AT(RTC_WUTR,0x5154);
513     /* Wakeup Timer Register High */
514     DEF_8BIT_REG_AT(RTC_WUTRH,0x5154);
515     /* Wakeup Timer Register Low */
516     DEF_8BIT_REG_AT(RTC_WUTRL,0x5155);
517
518     /* Subsecond register */
519     DEF_16BIT_REG_AT(RTC_SSR,0x5157);
520     /* Subsecond register low */
521     DEF_8BIT_REG_AT(RTC_SSRL,0x5157);
522     /* Subsecond register high */
523     DEF_8BIT_REG_AT(RTC_SSRH,0x5158);
524
525     /* Write Protection Register */
526     DEF_8BIT_REG_AT(RTC_WPR,0x5159);
527
528     /* Shift register */
529     DEF_16BIT_REG_AT(RTC_SHIFTR,0x515a);
530     /* Shift register high */
531     DEF_8BIT_REG_AT(RTC_SHIFTRH,0x515a);
532     /* Shift register low */
533     DEF_8BIT_REG_AT(RTC_SHIFTRL,0x515b);
534
535     /* Alarm A Register 1 */
536     DEF_8BIT_REG_AT(RTC_ALRMAR1,0x515c);
```

```
537
538     /* Alarm A Register 2 */
539     DEF_8BIT_REG_AT(RTC_ALRMAR2,0x515d);
540
541     /* Alarm A Register 3 */
542     DEF_8BIT_REG_AT(RTC_ALRMAR3,0x515e);
543
544     /* Alarm A Register 4 */
545     DEF_8BIT_REG_AT(RTC_ALRMAR4,0x515f);
546
547     /* Alarm A subsecond register */
548     DEF_16BIT_REG_AT(RTC_ALRMASSR,0x5164);
549     /* Alarm A subsecond register high */
550     DEF_8BIT_REG_AT(RTC_ALRMASSRH,0x5164);
551     /* Alarm A subsecond register low */
552     DEF_8BIT_REG_AT(RTC_ALRMASSRL,0x5165);
553
554     /* Alarm A masking register */
555     DEF_8BIT_REG_AT(RTC_ALRMASSMSKR,0x5166);
556
557     /* Calibration register */
558     DEF_16BIT_REG_AT(RTC_CALR,0x516a);
559     /* Calibration register high */
560     DEF_8BIT_REG_AT(RTC_CALRH,0x516a);
561     /* Calibration register low */
562     DEF_8BIT_REG_AT(RTC_CALRL,0x516b);
563
564     /* Tamper control register 1 */
565     DEF_8BIT_REG_AT(RTC_TCR1,0x516c);
566
567     /* Tamper control register 2 */
568     DEF_8BIT_REG_AT(RTC_TCR2,0x516d);
569
570     /*  (CSSLSE) */
571     /*****************************************************************/
572
573     /* CSS on LSE control and status register */
574     DEF_8BIT_REG_AT(CSSLSE_CSR,0x5190);
575
576     /* Serial Peripheral Interface 1 (SPI1) */
577     /*****************************************************************/
578
579     /* SPI1 Control Register 1 */
580     DEF_8BIT_REG_AT(SPI1_CR1,0x5200);
581
582     /* SPI1 Control Register 2 */
583     DEF_8BIT_REG_AT(SPI1_CR2,0x5201);
584
585     /* SPI1 Interrupt Control Register */
586     DEF_8BIT_REG_AT(SPI1_ICR,0x5202);
587
588     /* SPI1 Status Register */
589     DEF_8BIT_REG_AT(SPI1_SR,0x5203);
590
591     /* SPI1 Data Register */
592     DEF_8BIT_REG_AT(SPI1_DR,0x5204);
593
594     /* SPI1 CRC Polynomial Register */
595     DEF_8BIT_REG_AT(SPI1_CRCPR,0x5205);
596
597     /* SPI1 Rx CRC Register */
598     DEF_8BIT_REG_AT(SPI1_RXCRCR,0x5206);
599
600     /* SPI1 Tx CRC Register */
601     DEF_8BIT_REG_AT(SPI1_TXCRCR,0x5207);
602
603     /* I2C Bus Interface 1 (I2C1) */
```

```
/*****************************************************************/

/* I2C1 control register 1 */
DEF_8BIT_REG_AT(I2C1_CR1,0x5210);

/* I2C1 control register 2 */
DEF_8BIT_REG_AT(I2C1_CR2,0x5211);

/* I2C1 frequency register */
DEF_8BIT_REG_AT(I2C1_FREQR,0x5212);

/* I2C1 Own address register low */
DEF_8BIT_REG_AT(I2C1_OARL,0x5213);

/* I2C1 Own address register high */
DEF_8BIT_REG_AT(I2C1_OARH,0x5214);

/* I2C1 own address register for dual mode */
//DEF_8BIT_REG_AT(I2C1_OARH,0x5215);

/* I2C1 data register */
DEF_8BIT_REG_AT(I2C1_DR,0x5216);

/* I2C1 status register 1 */
DEF_8BIT_REG_AT(I2C1_SR1,0x5217);

/* I2C1 status register 2 */
DEF_8BIT_REG_AT(I2C1_SR2,0x5218);

/* I2C1 status register 3 */
DEF_8BIT_REG_AT(I2C1_SR3,0x5219);

/* I2C1 interrupt control register */
DEF_8BIT_REG_AT(I2C1_ITR,0x521a);

/* I2C1 Clock control register low */
DEF_8BIT_REG_AT(I2C1_CCRL,0x521b);

/* I2C1 Clock control register high */
DEF_8BIT_REG_AT(I2C1_CCRH,0x521c);

/* I2C1 TRISE register */
DEF_8BIT_REG_AT(I2C1_TRISER,0x521d);

/* I2C1 packet error checking register */
DEF_8BIT_REG_AT(I2C1_PECR,0x521e);

/* Universal synch/asynch receiver transmitter 1 (USART1) */
/*****************************************************************/

/* USART1 Status Register */
DEF_8BIT_REG_AT(USART1_SR,0x5230);

/* USART1 Data Register */
DEF_8BIT_REG_AT(USART1_DR,0x5231);

/* USART1 Baud Rate Register 1 */
DEF_8BIT_REG_AT(USART1_BRR1,0x5232);

/* USART1 Baud Rate Register 2 */
DEF_8BIT_REG_AT(USART1_BRR2,0x5233);

/* USART1 Control Register 1 */
DEF_8BIT_REG_AT(USART1_CR1,0x5234);

/* USART1 Control Register 2 */
DEF_8BIT_REG_AT(USART1_CR2,0x5235);
```

```
671
672     /* USART1 Control Register 3 */
673     DEF_8BIT_REG_AT(USART1_CR3,0x5236);
674
675     /* USART1 Control Register 4 */
676     DEF_8BIT_REG_AT(USART1_CR4,0x5237);
677
678     /* USART1 Control Register 5 */
679     DEF_8BIT_REG_AT(USART1_CR5,0x5238);
680
681     /* USART1 Guard time Register */
682     DEF_8BIT_REG_AT(USART1_GTR,0x5239);
683
684     /* USART1 Prescaler Register */
685     DEF_8BIT_REG_AT(USART1_PSCR,0x523a);
686
687     /* 16-Bit Timer 2 (TIM2) */
688     /*****************************************************************/
689
690     /* TIM2 Control register 1 */
691     DEF_8BIT_REG_AT(TIM2_CR1,0x5250);
692
693     /* TIM2 Control register 2 */
694     DEF_8BIT_REG_AT(TIM2_CR2,0x5251);
695
696     /* TIM2 Slave Mode Control register */
697     DEF_8BIT_REG_AT(TIM2_SMCR,0x5252);
698
699     /* TIM2 External trigger register */
700     DEF_8BIT_REG_AT(TIM2_ETR,0x5253);
701
702     /* TIM2 DMA request enable register */
703     DEF_8BIT_REG_AT(TIM2_DER,0x5254);
704
705     /* TIM2 Interrupt enable register */
706     DEF_8BIT_REG_AT(TIM2_IER,0x5255);
707
708     /* TIM2 Status register 1 */
709     DEF_8BIT_REG_AT(TIM2_SR1,0x5256);
710
711     /* TIM2 Status register 2 */
712     DEF_8BIT_REG_AT(TIM2_SR2,0x5257);
713
714     /* TIM2 Event Generation register */
715     DEF_8BIT_REG_AT(TIM2_EGR,0x5258);
716
717     /* TIM2 Capture/Compare mode register 1 */
718     DEF_8BIT_REG_AT(TIM2_CCMR1,0x5259);
719
720     /* TIM2 Capture/Compare mode register 2 */
721     DEF_8BIT_REG_AT(TIM2_CCMR2,0x525a);
722
723     /* TIM2 Capture/Compare enable register 1 */
724     DEF_8BIT_REG_AT(TIM2_CCER1,0x525b);
725
726     /* TIM2 Counter */
727     DEF_16BIT_REG_AT(TIM2_CNTR,0x525c);
728     /* TIM2 Counter High */
729     DEF_8BIT_REG_AT(TIM2_CNTRH,0x525c);
730     /* TIM2 Counter Low */
731     DEF_8BIT_REG_AT(TIM2_CNTRL,0x525d);
732
733     /* TIM2 Prescaler register */
734     DEF_8BIT_REG_AT(TIM2_PSCR,0x525e);
735
736     /* TIM2 Auto-reload register */
737     DEF_16BIT_REG_AT(TIM2_ARR,0x525f);
```

```
    /* TIM2 Auto-Reload Register High */
    DEF_8BIT_REG_AT(TIM2_ARRH,0x525f);
    /* TIM2 Auto-Reload Register Low */
    DEF_8BIT_REG_AT(TIM2_ARRL,0x5260);

    /* TIM2 Capture/Compare register 1 */
    DEF_16BIT_REG_AT(TIM2_CCR1,0x5261);
    /* TIM2 Capture/Compare Register 1 High */
    DEF_8BIT_REG_AT(TIM2_CCR1H,0x5261);
    /* TIM2 Capture/Compare Register 1 Low */
    DEF_8BIT_REG_AT(TIM2_CCR1L,0x5262);

    /* TIM2 Capture/Compare register 2 */
    DEF_16BIT_REG_AT(TIM2_CCR2,0x5263);
    /* TIM2 Capture/Compare Register 2 High */
    DEF_8BIT_REG_AT(TIM2_CCR2H,0x5263);
    /* TIM2 Capture/Compare Register 2 Low */
    DEF_8BIT_REG_AT(TIM2_CCR2L,0x5264);

    /* TIM2 Break register */
    DEF_8BIT_REG_AT(TIM2_BKR,0x5265);

    /* TIM2 Output idle state register */
    DEF_8BIT_REG_AT(TIM2_OISR,0x5266);

    /* 16-Bit Timer 3 (TIM3) */
    /*****************************************************************/

    /* TIM3 Control register 1 */
    DEF_8BIT_REG_AT(TIM3_CR1,0x5280);

    /* TIM3 Control register 2 */
    DEF_8BIT_REG_AT(TIM3_CR2,0x5281);

    /* TIM3 Slave Mode Control register */
    DEF_8BIT_REG_AT(TIM3_SMCR,0x5282);

    /* TIM3 External trigger register */
    DEF_8BIT_REG_AT(TIM3_ETR,0x5283);

    /* TIM3 DMA request enable register */
    DEF_8BIT_REG_AT(TIM3_DER,0x5284);

    /* TIM3 Interrupt enable register */
    DEF_8BIT_REG_AT(TIM3_IER,0x5285);

    /* TIM3 Status register 1 */
    DEF_8BIT_REG_AT(TIM3_SR1,0x5286);

    /* TIM3 Status register 2 */
    DEF_8BIT_REG_AT(TIM3_SR2,0x5287);

    /* TIM3 Event Generation register */
    DEF_8BIT_REG_AT(TIM3_EGR,0x5288);

    /* TIM3 Capture/Compare mode register 1 */
    DEF_8BIT_REG_AT(TIM3_CCMR1,0x5289);

    /* TIM3 Capture/Compare mode register 2 */
    DEF_8BIT_REG_AT(TIM3_CCMR2,0x528a);

    /* TIM3 Capture/Compare enable register 1 */
    DEF_8BIT_REG_AT(TIM3_CCER1,0x528b);

    /* TIM3 Counter */
    DEF_16BIT_REG_AT(TIM3_CNTR,0x528c);
    /* TIM3 Counter High */
```

```
    DEF_8BIT_REG_AT(TIM3_CNTRH,0x528c);
    /* TIM3 Counter Low */
    DEF_8BIT_REG_AT(TIM3_CNTRL,0x528d);

    /* TIM3 Prescaler register */
    DEF_8BIT_REG_AT(TIM3_PSCR,0x528e);

    /* TIM3 Auto-reload register */
    DEF_16BIT_REG_AT(TIM3_ARR,0x528f);
    /* TIM3 Auto-Reload Register High */
    DEF_8BIT_REG_AT(TIM3_ARRH,0x528f);
    /* TIM3 Auto-Reload Register Low */
    DEF_8BIT_REG_AT(TIM3_ARRL,0x5290);

    /* TIM3 Capture/Compare register 1 */
    DEF_16BIT_REG_AT(TIM3_CCR1,0x5291);
    /* TIM3 Capture/Compare Register 1 High */
    DEF_8BIT_REG_AT(TIM3_CCR1H,0x5291);
    /* TIM3 Capture/Compare Register 1 Low */
    DEF_8BIT_REG_AT(TIM3_CCR1L,0x5292);

    /* TIM3 Capture/Compare register 2 */
    DEF_16BIT_REG_AT(TIM3_CCR2,0x5293);
    /* TIM3 Capture/Compare Register 2 High */
    DEF_8BIT_REG_AT(TIM3_CCR2H,0x5293);
    /* TIM3 Capture/Compare Register 2 Low */
    DEF_8BIT_REG_AT(TIM3_CCR2L,0x5294);

    /* TIM3 Break register */
    DEF_8BIT_REG_AT(TIM3_BKR,0x5295);

    /* TIM3 Output idle state register */
    DEF_8BIT_REG_AT(TIM3_OISR,0x5296);

    /* 16-Bit Timer 1 (TIM1) */
    /*****************************************************************/

    /* TIM1 Control register 1 */
    DEF_8BIT_REG_AT(TIM1_CR1,0x52b0);

    /* TIM1 Control register 2 */
    DEF_8BIT_REG_AT(TIM1_CR2,0x52b1);

    /* TIM1 Slave Mode Control register */
    DEF_8BIT_REG_AT(TIM1_SMCR,0x52b2);

    /* TIM1 external trigger register */
    DEF_8BIT_REG_AT(TIM1_ETR,0x52b3);

    /* TIM1 DMA request enable register */
    DEF_8BIT_REG_AT(TIM1_DER,0x52b4);

    /* TIM1 Interrupt enable register */
    DEF_8BIT_REG_AT(TIM1_IER,0x52b5);

    /* TIM1 Status register 1 */
    DEF_8BIT_REG_AT(TIM1_SR1,0x52b6);

    /* TIM1 Status register 2 */
    DEF_8BIT_REG_AT(TIM1_SR2,0x52b7);

    /* TIM1 Event Generation register */
    DEF_8BIT_REG_AT(TIM1_EGR,0x52b8);

    /* TIM1 Capture/Compare mode register 1 */
    DEF_8BIT_REG_AT(TIM1_CCMR1,0x52b9);

```

```
872     /* TIM1 Capture/Compare mode register 2 */
873     DEF_8BIT_REG_AT(TIM1_CCMR2,0x52ba);
874
875     /* TIM1 Capture/Compare mode register 3 */
876     DEF_8BIT_REG_AT(TIM1_CCMR3,0x52bb);
877
878     /* TIM1 Capture/Compare mode register 4 */
879     DEF_8BIT_REG_AT(TIM1_CCMR4,0x52bc);
880
881     /* TIM1 Capture/Compare enable register 1 */
882     DEF_8BIT_REG_AT(TIM1_CCER1,0x52bd);
883
884     /* TIM1 Capture/Compare enable register 2 */
885     DEF_8BIT_REG_AT(TIM1_CCER2,0x52be);
886
887     /* TIM1 Counter */
888     DEF_16BIT_REG_AT(TIM1_CNTR,0x52bf);
889     /* TIM1 Counter High */
890     DEF_8BIT_REG_AT(TIM1_CNTRH,0x52bf);
891     /* TIM1 Counter Low */
892     DEF_8BIT_REG_AT(TIM1_CNTRL,0x52c0);
893
894     /* TIM1 Prescaler register */
895     DEF_16BIT_REG_AT(TIM1_PSCR,0x52c1);
896     /* TIM1 Prescaler Register High */
897     DEF_8BIT_REG_AT(TIM1_PSCRH,0x52c1);
898     /* TIM1 Prescaler Register Low */
899     DEF_8BIT_REG_AT(TIM1_PSCRL,0x52c2);
900
901     /* TIM1 Auto-reload register */
902     DEF_16BIT_REG_AT(TIM1_ARR,0x52c3);
903     /* TIM1 Auto-Reload Register High */
904     DEF_8BIT_REG_AT(TIM1_ARRH,0x52c3);
905     /* TIM1 Auto-Reload Register Low */
906     DEF_8BIT_REG_AT(TIM1_ARRL,0x52c4);
907
908     /* TIM1 Repetition counter register */
909     DEF_8BIT_REG_AT(TIM1_RCR,0x52c5);
910
911     /* TIM1 Capture/Compare register 1 */
912     DEF_16BIT_REG_AT(TIM1_CCR1,0x52c6);
913     /* TIM1 Capture/Compare Register 1 High */
914     DEF_8BIT_REG_AT(TIM1_CCR1H,0x52c6);
915     /* TIM1 Capture/Compare Register 1 Low */
916     DEF_8BIT_REG_AT(TIM1_CCR1L,0x52c7);
917
918     /* TIM1 Capture/Compare register 2 */
919     DEF_16BIT_REG_AT(TIM1_CCR2,0x52c8);
920     /* TIM1 Capture/Compare Register 2 High */
921     DEF_8BIT_REG_AT(TIM1_CCR2H,0x52c8);
922     /* TIM1 Capture/Compare Register 2 Low */
923     DEF_8BIT_REG_AT(TIM1_CCR2L,0x52c9);
924
925     /* TIM1 Capture/Compare register 3 */
926     DEF_16BIT_REG_AT(TIM1_CCR3,0x52ca);
927     /* TIM1 Capture/Compare Register 3 High */
928     DEF_8BIT_REG_AT(TIM1_CCR3H,0x52ca);
929     /* TIM1 Capture/Compare Register 3 Low */
930     DEF_8BIT_REG_AT(TIM1_CCR3L,0x52cb);
931
932     /* TIM1 Capture/Compare register 4 */
933     DEF_16BIT_REG_AT(TIM1_CCR4,0x52cc);
934     /* TIM1 Capture/Compare Register 4 High */
935     DEF_8BIT_REG_AT(TIM1_CCR4H,0x52cc);
936     /* TIM1 Capture/Compare Register 4 Low */
937     DEF_8BIT_REG_AT(TIM1_CCR4L,0x52cd);
938
```

```
    /* TIM1 Break register */
    DEF_8BIT_REG_AT(TIM1_BKR,0x52ce);

    /* TIM1 Dead-time register */
    DEF_8BIT_REG_AT(TIM1_DTR,0x52cf);

    /* TIM1 Output idle state register */
    DEF_8BIT_REG_AT(TIM1_OISR,0x52d0);

    /* TIM1 DMA control register 1 */
    DEF_8BIT_REG_AT(TIM1_DCR1,0x52d1);

    /* TIM1 DMA control register 2 */
    DEF_8BIT_REG_AT(TIM1_DCR2,0x52d2);

    /* TIM1 DMA address for burst mode */
    DEF_8BIT_REG_AT(TIM1_DMA1R,0x52d3);

    /* 8-Bit  Timer 4 (TIM4) */
    /*****************************************************************/

    /* TIM4 Control Register 1 */
    DEF_8BIT_REG_AT(TIM4_CR1,0x52e0);

    /* TIM4 Control Register 2 */
    DEF_8BIT_REG_AT(TIM4_CR2,0x52e1);

    /* TIM4 Slave Mode Control Register */
    DEF_8BIT_REG_AT(TIM4_SMCR,0x52e2);

    /* TIM4 DMA request Enable Register */
    DEF_8BIT_REG_AT(TIM4_DER,0x52e3);

    /* TIM4 Interrupt Enable Register */
    DEF_8BIT_REG_AT(TIM4_IER,0x52e4);

    /* TIM4 Status Register 1 */
    DEF_8BIT_REG_AT(TIM4_SR1,0x52e5);

    /* TIM4 Event Generation Register */
    DEF_8BIT_REG_AT(TIM4_EGR,0x52e6);

    /* TIM4 Counter */
    DEF_8BIT_REG_AT(TIM4_CNTR,0x52e7);

    /* TIM4 Prescaler Register */
    DEF_8BIT_REG_AT(TIM4_PSCR,0x52e8);

    /* TIM4 Auto-Reload Register */
    DEF_8BIT_REG_AT(TIM4_ARR,0x52e9);

    /* Infra Red Interface (IRTIM) */
    /*****************************************************************/

    /* Infra-red control register */
    DEF_8BIT_REG_AT(IR_CR,0x52ff);

    /* 16-Bit Timer 5 (TIM5) */
    /*****************************************************************/

    /* TIM5 Control register 1 */
    DEF_8BIT_REG_AT(TIM5_CR1,0x5300);

    /* TIM5 Control register 2 */
    DEF_8BIT_REG_AT(TIM5_CR2,0x5301);

    /* TIM5 Slave Mode Control register */
```

```
    DEF_8BIT_REG_AT(TIM5_SMCR,0x5302);

    /* TIM5 external trigger register */
    DEF_8BIT_REG_AT(TIM5_ETR,0x5303);

    /* TIM5 DMA request enable register */
    DEF_8BIT_REG_AT(TIM5_DER,0x5304);

    /* TIM5 Interrupt enable register */
    DEF_8BIT_REG_AT(TIM5_IER,0x5305);

    /* TIM5 Status register 1 */
    DEF_8BIT_REG_AT(TIM5_SR1,0x5306);

    /* TIM5 Status register 2 */
    DEF_8BIT_REG_AT(TIM5_SR2,0x5307);

    /* TIM5 Event Generation register */
    DEF_8BIT_REG_AT(TIM5_EGR,0x5308);

    /* TIM5 Capture/Compare mode register 1 */
    DEF_8BIT_REG_AT(TIM5_CCMR1,0x5309);

    /* TIM5 Capture/Compare mode register 2 */
    DEF_8BIT_REG_AT(TIM5_CCMR2,0x530a);

    /* TIM5 Capture/Compare enable register 1 */
    DEF_8BIT_REG_AT(TIM5_CCER1,0x530b);

    /* TIM5 Counter */
    DEF_16BIT_REG_AT(TIM5_CNTR,0x530c);
    /* TIM5 Counter High */
    DEF_8BIT_REG_AT(TIM5_CNTRH,0x530c);
    /* TIM5 Counter Low */
    DEF_8BIT_REG_AT(TIM5_CNTRL,0x530d);

    /* TIM5 Prescaler register */
    DEF_8BIT_REG_AT(TIM5_PSCR,0x530e);


    /* TIM5 Auto-reload register */
    DEF_16BIT_REG_AT(TIM5_ARR,0x530f);
    /* TIM5 Auto-Reload Register High */
    DEF_8BIT_REG_AT(TIM5_ARRH,0x530f);
    /* TIM5 Auto-Reload Register Low */
    DEF_8BIT_REG_AT(TIM5_ARRL,0x5310);

    /* TIM5 Capture/Compare register 1 */
    DEF_16BIT_REG_AT(TIM5_CCR1,0x5311);
    /* TIM5 Capture/Compare Register 1 High */
    DEF_8BIT_REG_AT(TIM5_CCR1H,0x5311);
    /* TIM5 Capture/Compare Register 1 Low */
    DEF_8BIT_REG_AT(TIM5_CCR1L,0x5312);

    /* TIM5 Capture/Compare register 2 */
    DEF_16BIT_REG_AT(TIM5_CCR2,0x5313);
    /* TIM5 Capture/Compare Register 2 High */
    DEF_8BIT_REG_AT(TIM5_CCR2H,0x5313);
    /* TIM5 Capture/Compare Register 2 Low */
    DEF_8BIT_REG_AT(TIM5_CCR2L,0x5314);

    /* TIM5 Break register */
    DEF_8BIT_REG_AT(TIM5_BKR,0x5315);

    /* TIM5 Output idle state register */
    DEF_8BIT_REG_AT(TIM5_OISR,0x5316);

```

```
    /* Analog to digital converter 1 (ADC1) */
    /*****************************************************************/

    /* ADC1 Configuration register 1 */
    DEF_8BIT_REG_AT(ADC1_CR1,0x5340);

    /* ADC1 Configuration register 2 */
    DEF_8BIT_REG_AT(ADC1_CR2,0x5341);

    /* ADC1 Configuration register 3 */
    DEF_8BIT_REG_AT(ADC1_CR3,0x5342);

    /* ADC1 status register */
    DEF_8BIT_REG_AT(ADC1_SR,0x5343);

    /* ADC1 Data register */
    DEF_16BIT_REG_AT(ADC1_DR,0x5344);
    /* ADC Data Register High */
    DEF_8BIT_REG_AT(ADC1_DRH,0x5344);
    /* ADC Data Register Low */
    DEF_8BIT_REG_AT(ADC1_DRL,0x5345);

    /* ADC1 high threshold register */
    DEF_16BIT_REG_AT(ADC1_HTR,0x5346);
    /* ADC High Threshold Register High */
    DEF_8BIT_REG_AT(ADC1_HTRH,0x5346);
    /* ADC High Threshold Register Low */
    DEF_8BIT_REG_AT(ADC1_HTRL,0x5347);

    /* ADC1 low threshold register */
    DEF_16BIT_REG_AT(ADC1_LTR,0x5348);
    /* ADC Low Threshold Register High */
    DEF_8BIT_REG_AT(ADC1_LTRH,0x5348);
    /* ADC Low Threshold Register Low */
    DEF_8BIT_REG_AT(ADC1_LTRL,0x5349);

    /* ADC1 channel sequence 1 register */
    DEF_8BIT_REG_AT(ADC1_SQR1,0x534a);

    /* ADC1 channel sequence 2 register */
    DEF_8BIT_REG_AT(ADC1_SQR2,0x534b);

    /* ADC1 channel sequence 3 register */
    DEF_8BIT_REG_AT(ADC1_SQR3,0x534c);

    /* ADC1 channel sequence 4 register */
    DEF_8BIT_REG_AT(ADC1_SQR4,0x534d);

    /* ADC1 Trigger disable 1 */
    DEF_8BIT_REG_AT(ADC1_TRIGR1,0x534e);

    /* ADC1 Trigger disable 2 */
    DEF_8BIT_REG_AT(ADC1_TRIGR2,0x534f);

    /* ADC1 Trigger disable 3 */
    DEF_8BIT_REG_AT(ADC1_TRIGR3,0x5350);

    /* ADC1 Trigger disable 4 */
    DEF_8BIT_REG_AT(ADC1_TRIGR4,0x5351);

    /* Digital to analog converter (DAC) */
    /*****************************************************************/

    /* DAC channel 1 control register 1 */
    DEF_8BIT_REG_AT(DAC_CH1CR1,0x5380);

    /* DAC channel 1 control register 2 */
```

```
    DEF_8BIT_REG_AT(DAC_CH1CR2,0x5381);

    /* DAC channel 2 control register 1 */
    DEF_8BIT_REG_AT(DAC_CH2CR1,0x5382);

    /* DAC channel 2 control register 2 */
    DEF_8BIT_REG_AT(DAC_CH2CR2,0x5383);

    /* DAC software trigger register */
    DEF_8BIT_REG_AT(DAC_SWTRIGR,0x5384);

    /* DAC status register */
    DEF_8BIT_REG_AT(DAC_SR,0x5385);

    /* DAC channel 1 right aligned data holding register */
    DEF_16BIT_REG_AT(DAC_CH1RDHR,0x5388);
    /* DAC channel 1 right aligned data holding register high */
    DEF_8BIT_REG_AT(DAC_CH1RDHRH,0x5388);
    /* DAC channel 1 right aligned data holding register low */
    DEF_8BIT_REG_AT(DAC_CH1RDHRL,0x5389);

    /* DAC channel 1 left aligned data holding register */
    DEF_16BIT_REG_AT(DAC_CH1LDHR,0x538c);
    /* DAC channel 1 left aligned data holding register high */
    DEF_8BIT_REG_AT(DAC_CH1LDHRH,0x538c);
    /* DAC channel 1 left aligned data holding register low */
    DEF_8BIT_REG_AT(DAC_CH1LDHRL,0x538d);

    /* DAC channel 1 8-bit data holding register */
    DEF_8BIT_REG_AT(DAC_CH1DHR8,0x5390);

    /* DAC channel 2 right aligned data holding register */
    DEF_16BIT_REG_AT(DAC_CH2RDHR,0x5394);
    /* DAC channel 2 right aligned data holding register high */
    DEF_8BIT_REG_AT(DAC_CH2RDHRH,0x5394);
    /* DAC channel 2 right aligned data holding register low */
    DEF_8BIT_REG_AT(DAC_CH2RDHRL,0x5395);

    /* DAC channel 2 left aligned data holding register */
    DEF_16BIT_REG_AT(DAC_CH2LDHR,0x5398);
    /* DAC channel 2 left aligned data holding register high */
    DEF_8BIT_REG_AT(DAC_CH2LDHRH,0x5398);
    /* DAC channel 2 left aligned data holding register low */
    DEF_8BIT_REG_AT(DAC_CH2LDHRL,0x5399);

    /* DAC channel 2 8-bit data holding register */
    DEF_8BIT_REG_AT(DAC_CH2DHR8,0x539c);

    /* DAC channel 1 right aligned data holding register */
    DEF_16BIT_REG_AT(DAC_DCH1RDHR,0x53a0);
    /* DAC channel 1 right aligned data holding register high */
    DEF_8BIT_REG_AT(DAC_DCH1RDHRH,0x53a0);
    /* DAC channel 1 right aligned data holding register low */
    DEF_8BIT_REG_AT(DAC_DCH1RDHRL,0x53a1);

    /* DAC data output register */
    DEF_16BIT_REG_AT(DAC_DOR,0x53ac);
    /* DAC data output register high */
    DEF_8BIT_REG_AT(DAC_DORH,0x53ac);
    /* DAC data output register low */
    DEF_8BIT_REG_AT(DAC_DORL,0x53ad);

    /* DAC channel 2 right aligned data holding register */
    DEF_16BIT_REG_AT(DAC_DCH2RDHR,0x53a2);
    /* DAC channel 2 right aligned data holding register high */
    DEF_8BIT_REG_AT(DAC_DCH2RDHRH,0x53a2);
    /* DAC channel 2 right aligned data holding register low */
```

```
    DEF_8BIT_REG_AT(DAC_DCH2RDHRL,0x53a3);

    /* DAC channel 1left aligned data holding register */
    DEF_16BIT_REG_AT(DAC_DCH1LDHR,0x53a4);
    /* DAC channel 1left aligned data holding register high */
    DEF_8BIT_REG_AT(DAC_DCH1LDHRH,0x53a4);
    /* DAC channel 1left aligned data holding register low */
    DEF_8BIT_REG_AT(DAC_DCH1LDHRL,0x53a5);

    /* DAC channel 2 left aligned data holding register */
    DEF_16BIT_REG_AT(DAC_DCH2LDHR,0x53a6);
    /* DAC channel 2 left aligned data holding register high */
    DEF_8BIT_REG_AT(DAC_DCH2LDHRH,0x53a6);
    /* DAC channel 2 left aligned data holding register low */
    DEF_8BIT_REG_AT(DAC_DCH2LDHRL,0x53a7);

    /* DAC channel 1 8-bit mode data holding register */
    DEF_8BIT_REG_AT(DAC_DCH1DHR8,0x53a8);

    /* DAC channel 2 8-bit mode data holding register */
    DEF_8BIT_REG_AT(DAC_DCH2DHR8,0x53a9);

    /* DAC channel 1 data output register */
    DEF_16BIT_REG_AT(DAC_CH1DOR,0x53ac);
    /* DAC channel 1 data output register high */
    DEF_8BIT_REG_AT(DAC_CH1DORH,0x53ac);
    /* DAC channel 1 data output register low */
    DEF_8BIT_REG_AT(DAC_CH1DORL,0x53ad);

    /* DAC channel 2 data output register */
    DEF_16BIT_REG_AT(DAC_CH2DOR,0x53b0);
    /* DAC channel 2 data output register high */
    DEF_8BIT_REG_AT(DAC_CH2DORH,0x53b0);
    /* DAC channel 2 data output register low */
    DEF_8BIT_REG_AT(DAC_CH2DORL,0x53b1);

    /* Serial Peripheral Interface 2 (SPI2) */
    /*****************************************************************/

    /* SPI2 Control Register 1 */
    DEF_8BIT_REG_AT(SPI2_CR1,0x53c0);

    /* SPI2 Control Register 2 */
    DEF_8BIT_REG_AT(SPI2_CR2,0x53c1);

    /* SPI2 Interrupt Control Register */
    DEF_8BIT_REG_AT(SPI2_ICR,0x53c2);

    /* SPI2 Status Register */
    DEF_8BIT_REG_AT(SPI2_SR,0x53c3);

    /* SPI2 Data Register */
    DEF_8BIT_REG_AT(SPI2_DR,0x53c4);

    /* SPI2 CRC Polynomial Register */
    DEF_8BIT_REG_AT(SPI2_CRCPR,0x53c5);

    /* SPI2 Rx CRC Register */
    DEF_8BIT_REG_AT(SPI2_RXCRCR,0x53c6);

    /* SPI2 Tx CRC Register */
    DEF_8BIT_REG_AT(SPI2_TXCRCR,0x53c7);

    /* Universal synch/asynch receiver transmitter 2 (USART2) */
    /*****************************************************************/

    /* USART2 Status Register */
```

```
    DEF_8BIT_REG_AT(USART2_SR,0x53e0);

    /* USART2 Data Register */
    DEF_8BIT_REG_AT(USART2_DR,0x53e1);

    /* USART2 Baud Rate Register 1 */
    DEF_8BIT_REG_AT(USART2_BRR1,0x53e2);

    /* USART2 Baud Rate Register 2 */
    DEF_8BIT_REG_AT(USART2_BRR2,0x53e3);

    /* USART2 Control Register 1 */
    DEF_8BIT_REG_AT(USART2_CR1,0x53e4);

    /* USART2 Control Register 2 */
    DEF_8BIT_REG_AT(USART2_CR2,0x53e5);

    /* USART2 Control Register 3 */
    DEF_8BIT_REG_AT(USART2_CR3,0x53e6);

    /* USART2 Control Register 4 */
    DEF_8BIT_REG_AT(USART2_CR4,0x53e7);

    /* USART2 Control Register 5 */
    DEF_8BIT_REG_AT(USART2_CR5,0x53e8);

    /* USART2 Guard time Register */
    DEF_8BIT_REG_AT(USART2_GTR,0x53e9);

    /* USART2 Prescaler Register */
    DEF_8BIT_REG_AT(USART2_PSCR,0x53ea);

    /* Universal synch/asynch receiver transmitter 3 (USART3) */
    /*****************************************************************/

    /* USART3 Status Register */
    DEF_8BIT_REG_AT(USART3_SR,0x53f0);

    /* USART3 Data Register */
    DEF_8BIT_REG_AT(USART3_DR,0x53f1);

    /* USART3 Baud Rate Register 1 */
    DEF_8BIT_REG_AT(USART3_BRR1,0x53f2);

    /* USART3 Baud Rate Register 2 */
    DEF_8BIT_REG_AT(USART3_BRR2,0x53f3);

    /* USART3 Control Register 1 */
    DEF_8BIT_REG_AT(USART3_CR1,0x53f4);

    /* USART3 Control Register 2 */
    DEF_8BIT_REG_AT(USART3_CR2,0x53f5);

    /* USART3 Control Register 3 */
    DEF_8BIT_REG_AT(USART3_CR3,0x53f6);

    /* USART3 Control Register 4 */
    DEF_8BIT_REG_AT(USART3_CR4,0x53f7);

    /* USART3 Control Register 5 */
    DEF_8BIT_REG_AT(USART3_CR5,0x53f8);

    /* USART3 Guard time Register */
    DEF_8BIT_REG_AT(USART3_GTR,0x53f9);

    /* USART3 Prescaler Register */
    DEF_8BIT_REG_AT(USART3_PSCR,0x53fa);
```

```
1341
1342    /* Routing interface (RI) */
1343    /*****************************************************************/
1344
1345    /* Timer input capture routing register 1 */
1346    DEF_8BIT_REG_AT(RI_ICR1,0x5431);
1347
1348    /* Timer input capture routing register 2 */
1349    DEF_8BIT_REG_AT(RI_ICR2,0x5432);
1350
1351    /* I/O input register 1 */
1352    DEF_8BIT_REG_AT(RI_IOIR1,0x5433);
1353
1354    /* I/O input register 2 */
1355    DEF_8BIT_REG_AT(RI_IOIR2,0x5434);
1356
1357    /* I/O input register 3 */
1358    DEF_8BIT_REG_AT(RI_IOIR3,0x5435);
1359
1360    /* I/O control mode register 1 */
1361    DEF_8BIT_REG_AT(RI_IOCMR1,0x5436);
1362
1363    /* I/O control mode register 2 */
1364    DEF_8BIT_REG_AT(RI_IOCMR2,0x5437);
1365
1366    /* I/O control mode register 3 */
1367    DEF_8BIT_REG_AT(RI_IOCMR3,0x5438);
1368
1369    /* I/O switch register 1 */
1370    DEF_8BIT_REG_AT(RI_IOSR1,0x5439);
1371
1372    /* I/O switch register 2 */
1373    DEF_8BIT_REG_AT(RI_IOSR2,0x543a);
1374
1375    /* I/O switch register 3 */
1376    DEF_8BIT_REG_AT(RI_IOSR3,0x543b);
1377
1378    /* I/O group control register */
1379    DEF_8BIT_REG_AT(RI_IOGCR,0x543c);
1380
1381    /* Analog switch register 1 */
1382    DEF_8BIT_REG_AT(RI_ASCR1,0x543d);
1383
1384    /* Analog switch register 2 */
1385    DEF_8BIT_REG_AT(RI_ASCR2,0x543e);
1386
1387    /* Resistor control register 1 */
1388    DEF_8BIT_REG_AT(RI_RCR,0x543f);
1389
1390    /* Comparators (COMP) */
1391    /*****************************************************************/
1392
1393    /* Comparator control and status register 1 */
1394    DEF_8BIT_REG_AT(COMP_CSR1,0x5440);
1395
1396    /* Comparator control and status register 2 */
1397    DEF_8BIT_REG_AT(COMP_CSR2,0x5441);
1398
1399    /* Comparator control and status register 3 */
1400    DEF_8BIT_REG_AT(COMP_CSR3,0x5442);
1401
1402    /* Comparator control and status register 4 */
1403    DEF_8BIT_REG_AT(COMP_CSR4,0x5443);
1404
1405    /* Comparator control and status register 5 */
1406    DEF_8BIT_REG_AT(COMP_CSR5,0x5444);
1407
```

```
1408    /*  (CPU) */
1409    /*****************************************************************/
1410
1411    /* Accumulator */
1412    DEF_8BIT_REG_AT(A,0x7f00);
1413
1414    /* Program counter extended */
1415    DEF_8BIT_REG_AT(PCE,0x7f01);
1416
1417    /* Program counter high */
1418    DEF_8BIT_REG_AT(PCH,0x7f02);
1419
1420    /* Program counter low */
1421    DEF_8BIT_REG_AT(PCL,0x7f03);
1422
1423    /* X index register high */
1424    DEF_8BIT_REG_AT(XH,0x7f04);
1425
1426    /* X index register low */
1427    DEF_8BIT_REG_AT(XL,0x7f05);
1428
1429    /* Y index register high */
1430    DEF_8BIT_REG_AT(YH,0x7f06);
1431
1432    /* Y index register low */
1433    DEF_8BIT_REG_AT(YL,0x7f07);
1434
1435    /* Stack pointer high */
1436    DEF_8BIT_REG_AT(SPH,0x7f08);
1437
1438    /* Stack pointer low */
1439    DEF_8BIT_REG_AT(SPL,0x7f09);
1440
1441    /* Condition code register */
1442    DEF_8BIT_REG_AT(CCR,0x7f0a);
1443
1444    /* Global configuration register (CFG) */
1445    /*****************************************************************/
1446
1447    /* CFG Global configuration register */
1448    DEF_8BIT_REG_AT(CFG_GCR,0x7f60);
1449
1450    /* Interrupt Software Priority Registers (ITC) */
1451    /*****************************************************************/
1452
1453    /* Interrupt Software priority register 1 */
1454    DEF_8BIT_REG_AT(ITC_SPR1,0x7f70);
1455
1456    /* Interrupt Software priority register 2 */
1457    DEF_8BIT_REG_AT(ITC_SPR2,0x7f71);
1458
1459    /* Interrupt Software priority register 3 */
1460    DEF_8BIT_REG_AT(ITC_SPR3,0x7f72);
1461
1462    /* Interrupt Software priority register 4 */
1463    DEF_8BIT_REG_AT(ITC_SPR4,0x7f73);
1464
1465    /* Interrupt Software priority register 5 */
1466    DEF_8BIT_REG_AT(ITC_SPR5,0x7f74);
1467
1468    /* Interrupt Software priority register 6 */
1469    DEF_8BIT_REG_AT(ITC_SPR6,0x7f75);
1470
1471    /* Interrupt Software priority register 7 */
1472    DEF_8BIT_REG_AT(ITC_SPR7,0x7f76);
1473
1474    /* Interrupt Software priority register 8 */
```

```
1475     DEF_8BIT_REG_AT(ITC_SPR8,0x7f77);
1476
1477
1478     /* (SWIM) */
1479     /*****************************************************************/
1480
1481     /* SWIM control status register */
1482     DEF_8BIT_REG_AT(SWIM_CSR,0x7f80);
1483
1484     #endif /* __STM8L151__ */
1485
```

```
/* STM8L151R8.h */
#ifdef MCU_NAME
#define STM8L151R8 1
#endif
#include "STM8L151x8.h"
```

```c
/*  MATHEMATICAL FUNCTIONS HEADER
 *  Copyright (c) 2006 by COSMIC Software
 */
#ifndef __MATH__
#define __MATH__  1

#define HUGE_VAL 1E38

double acos(double x);
double asin(double x);
double atan(double x);
double atan2(double y, double x);
double ceil(double x);
double cos(double x);
double cosh(double x);
double exp(double x);
double fabs(double x);
double floor(double x);
double fmod(double x, double y);
double frexp(double x, int *pexp);
double ldexp(double x, int exp);
double log(double x);
double log10(double x);
double modf(double value, double *pd);
double pow(double x, double y);
double sin(double x);
double sinh(double x);
double sqrt(double x);
double tan(double x);
double tanh(double x);

#endif

```

```c
/*  STRING TYPES HEADER
 *  Copyright (c) 2006 by COSMIC Software
 */

#ifndef __STRING__
#define __STRING__  1

#ifndef NULL
#define NULL  (void *)0
#endif

#ifdef NOINLINE
#define _INLINE
#else
#define _INLINE @inline
#endif

/*  function declarations
 */
char *strcat(char *s1, char *s2);
char *strchr(char *s, char c);
char * _INLINE strcpy(char *s1, char *s2);
char *strncat(char *s1, char *s2, unsigned int n);
char *strncpy(char *s1, char *s2, unsigned int n);
char *strpbrk(char *s1, char *s2);
char *strrchr(char *s, char c);
char *strstr(char *s1, char *s2);
int memcmp(void *s1, void *s2, unsigned int n);
int strcmp(char *s1, char *s2);
int strncmp(char *s1, char *s2, unsigned int n);
unsigned int strcspn(char *s1, char *s2);
_INLINE unsigned int strlen(char *s);
unsigned int strspn(char *s1, char *s2);
void *memchr(void *s, char c, unsigned int n);
void * _INLINE memcpy(void *s1, void *s2, unsigned int n);
void *memmove(void *s1, void *s2, unsigned int n);
void * _INLINE memset(void *s, char c, unsigned int n);

void eepcpy(@eeprom void *s1, void *s2, unsigned int n);
void eepset(@eeprom void *s1, char c, unsigned int n);
#define eepera(s1, n) (eepset(s1, 0, n))

#endif

```

HMC6300

HMC6300	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	BIN data	HEX data	HEX row
ROW0	0	0	0	0	0	0	0	0	00000000	00	00
ROW1	1	1	0	0	1	0	1	0	11001010	CA	01
ROW2	1	1	1	1	1	1	1	1	11111111	FF	02
ROW3	1	1	1	1	0	1	1	0	11110110	F6	03
ROW4	0	0	0	0	0	0	0	0	00000000	00	04
ROW5	1	1	1	1	1	1	1	1	11111111	FF	05
ROW6	1	1	1	0	1	1	0	0	11101100	EC	06
ROW7	0	0	0	0	1	1	1	1	00001111	0F	07
ROW8	1	0	0	0	1	1	1	1	10001111	8F	08
ROW9	0	0	0	0	0	0	0	0	00000000	00	09
ROW10	0	1	0	1	0	0	0	1	01010001	51	0A
ROW11	0	0	0	0	0	0	1	1	00000011	03	0B
ROW12	0	1	1	0	0	1	0	0	01100100	64	0C
ROW13	0	0	0	0	0	0	0	0	00000000	00	0D
ROW14	0	0	0	0	0	0	0	0	00000000	00	0E
ROW15	0	0	0	0	0	0	0	0	00000000	00	0F
ROW16	0	0	1	1	0	1	1	0	00110110	36	10
ROW17	1	0	1	1	1	0	1	1	10111011	BB	11
ROW18	0	1	0	0	0	1	1	0	01000110	46	12
ROW19	0	0	0	0	0	0	1	0	00000010	02	13
ROW20	0	0	1	1	0	1	0	1	00110101	35	14
ROW21	0	0	0	1	0	0	1	0	00010010	12	15
ROW22	0	0	0	0	1	0	1	0	00001010	0A	16
ROW23	0	1	1	0	0	0	1	0	01100010	62	17
ROW24	0	0	0	0	R	R	R	R			18
ROW25	R	R	R	R	R	R	R	R			19
ROW26	R	R	R	R	R	R	R	R			1A
ROW27	0	0	0	R	R	R	R	R			1B
ROW28	0	0	0	0	0	0	0	0	00000000	00	1C
ROW29	0	0	0	0	0	0	0	0	00000000	00	1D
ROW30	0	0	0	0	0	0	0	0	00000000	00	1E

HMC6301

HMC6301	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	BIN data	HEX data	HEX row
ROW0	0	0	1	0	0	0	0	0	00100000	20	00
ROW1	0	1	0	0	1	1	0	0	01001100	4C	01
ROW2	0	0	0	0	0	0	1	1	00000011	03	02
ROW3	0	0	0	0	0	0	1	1	00000011	03	03
ROW4	1	0	0	1	0	0	1	1	10010011	93	04
ROW5	1	1	1	1	1	1	1	1	11111111	FF	05
ROW6	1	0	1	1	1	1	1	1	10111111	BF	06
ROW7	0	1	1	0	1	1	0	1	01101101	6D	07
ROW8	1	0	0	0	0	0	0	0	10000000	80	08
ROW9	0	1	0	0	0	0	0	0	01000000	40	09
ROW10	0	0	0	0	0	0	0	0	00000000	00	0A
ROW11	0	0	0	0	0	0	0	0	00000000	00	0B
ROW12	0	0	0	0	0	0	0	0	00000000	00	0C
ROW13	0	0	0	0	0	0	0	0	00000000	00	0D
ROW14	0	0	0	0	0	0	0	0	00000000	00	0E
ROW15	0	0	0	0	0	0	0	0	00000000	00	0F
ROW16	0	0	1	1	0	1	1	0	00110110	36	10
ROW17	1	0	1	1	1	0	1	1	10111011	BB	11
ROW18	0	1	0	0	0	1	1	0	01000110	46	12
ROW19	0	0	0	0	0	0	1	0	00000010	02	13
ROW20	0	0	1	0	1	0	1	1	00101011	2B	14
ROW21	0	0	0	1	0	0	1	0	00010010	12	15
ROW22	0	0	0	0	0	1	0	1	00000101	05	16
ROW23	0	1	1	0	0	0	1	0	01100010	62	17
ROW24	0	0	0	0	R	R	R	R			18
ROW25	R	R	R	R	R	R	R	R			19
ROW26	R	R	R	R	R	R	R	R			1A
ROW27	0	0	0	R	R	R	R	R			1B
ROW28	0	0	0	0	0	0	0	0	00000000	00	1C
ROW29	0	0	0	0	0	0	0	0	00000000	00	1D
ROW30	0	0	0	0	0	0	0	0	00000000	00	1E

AD9833

AD9833	**MCLK [MHz]**	**FOUT [Hz]**	**FreqReg [DEC]**	**FreqReg [HEX]**	**FreqReg [BIN]**	
Calculations	25	1843621.4	19795734.0480143	12E0F16	00010010111000001111000101 10	
	Name		BIN code	Description		HEX code
Commands	Control Register		0010000100000000	DB13 & RESET		2100
	FreqReg 0 LSB		0100111100010110	DB14 & DB15 = 01 for FreqReg0 + 14LSBs of data		4F16
	FreqReg 0 MSB		0100010010111000	DB14 & DB15 = 01 for FreqReg0 + 14MSBs of data		44B8
	PhasReg 0		1100000000000000	DB14 & DB15 & DB13 = 110 for PhasReg0 + 12 MSBs of data		C000
	Exit Reset		0010000000000000	Reset set to 0 -> output after 7 MCLK		2000

MAX2831

Register	A3:A0	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	Function	BIN	HEX
0	0000	0	0	0	1	1	1	0	1	0	0	0	0	0	0	PLL mode	00011101000000	740
1	0001	0	1	0	0	0	1	1	0	0	1	1	0	1	0	Lock-Detect Output Select	01000110011010	119A
2	0010	0	1	0	0	0	0	0	0	0	0	0	0	1	1	Recommended values	01000000000011	1003
3	0011	1	1	0	0	1	1	0	1	1	1	1	0	1	0	F/N Main Divider	11001101111010	337A
4	0100	0	1	1	1	1	1	1	1	1	1	0	1	1	1	F Main Divider	01111111110111	1FF7
5	0101	0	0	0	0	0	0	1	0	1	0	0	1	0	0	LD enable & R Divider	00000010100100	A4
6	0110	0	0	0	0	0	0	0	0	1	0	0	0	0	0	Tx/Rx Calibration Mode	00000000100000	20
7	0111	0	1	0	0	0	0	0	0	1	0	0	0	1	0	Tx/Rx HP/LP Corner Frequencies	01000000100010	1022
8	1000	1	1	0	1	0	0	0	0	1	0	0	0	0	1	SPI, RSSI and LPF	11010000100001	3421
9	1001	0	0	0	1	1	1	1	0	1	1	0	1	0	1	Enable SPI Programming	00011110110101	7B5
10	1010	0	1	1	1	0	1	1	0	1	0	0	1	0	0	First/Second Stage PA	01110110100100	1DA4
11	1011	0	0	0	0	0	0	0	1	1	1	1	1	1	1	LNA/VGA Gain Control Settings	00000001111111	7F
12	1100	0	0	0	0	0	1	0	1	0	0	0	0	0	0	Tx VGA Gain Control	00000101000000	140
13	1101	0	0	1	1	1	0	1	0	0	1	0	0	1	0	Recommended values	00111010010010	E92
14	1110	0	0	0	0	0	1	0	0	1	1	1	0	1	1	Ref Clock Output/Crystal Fine Tune	00000100111011	13B
15	1111	0	0	0	1	0	1	0	1	0	0	0	1	0	1	Receiver I/Q Output Voltage	00010101000101	545

MAX2831 Frequency calculation

ONLY CHANGE LO, REF and R	
	MAX2830/31/32
LO	2449.99
VCO	2449.99
REF	40
R	2
PFD	20
N	122
F	0.4995
N in binary	01111010
F in binary	01111111110111110011

Modes of operation

Mode	SHDN	RXTN	Rx Path	Tx Path	PLL, VCO, LO GEN, AUTO-TUNER
Shutdown	0	0	off	off	off
Standby	0	1	off	off	on
Rx	1	0	on	off	on
Tx	1	1	off	on	on
Rx Calibration	1	0	on	upconv	on
Tx Calibration	1	1	off	on	on

ADC Channel Sequence

	BIN	HEX
ADC_SQR4	00000000	0
ADC_SQR3	00000000	0
ADC_SQR2	00000000	0
ADC_SQR1	11110	1E

DRS PCB Debug

Marcel Balle

October 9, 2018

Student:	Marcel Balle	.....
Lab Director:	Prof. Lixin Ran	.....
Thesis Director:	Dr. Alexandra Andersson	.....
Date:	October 9, 2018	

Contents

1 Introduction

1.1 Description

This document describes all the necessary debug tests performed on the PCB design. Each part is evaluated through measurements. These tests are executed in a specific order to ensure the proper operations for every device. The chips are mounted on the board step by step to avoid damaging any components in case some errors occur.

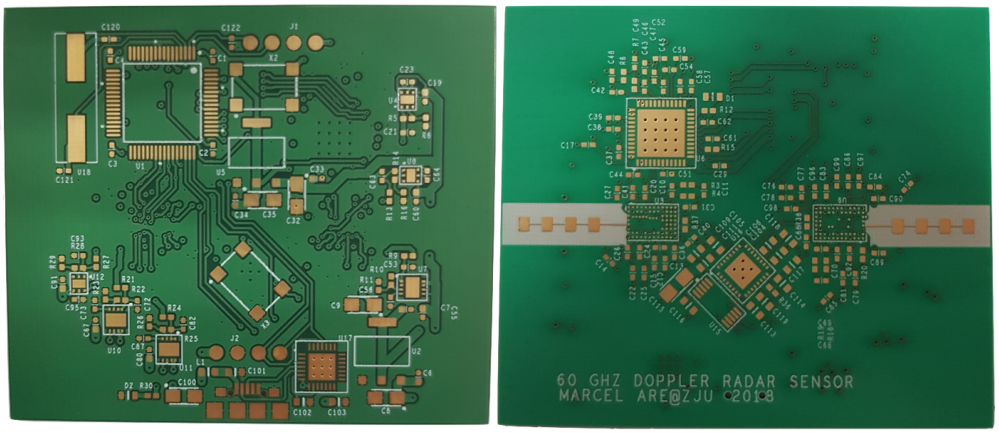


Figure 1: Blank PCB top and bottom

1.2 List of material

In order to debug the board and software correctly, some equipment is needed to execute all the measurements and tests described in this document. Apart from the sensor PCB board, including all the components listed in the bill of materials, the equipment used is specified the list below.

- Micro USB cable	
- PC for main power supply	
- SWIM connector	ST-Link V2
- Oscilloscope	Tektronix TDS 3034C
- 2 probes	Agilent 10073C
- Multimeter	VICTOR VC97
- DC power supply	Tektronix PWS2323-SC

2 Power supply

2.1 Common GND

Before soldering any components onto the board, a test is done to ensure no short circuits are present between the voltage supplies and the GND on the entire PCB.

2.2 Micro USB

The first step is to make sure the main power supply works properly. For this, the micro USB connector is mounted on the board along with its decoupling capacitors and the LED indicating the presence of voltage.

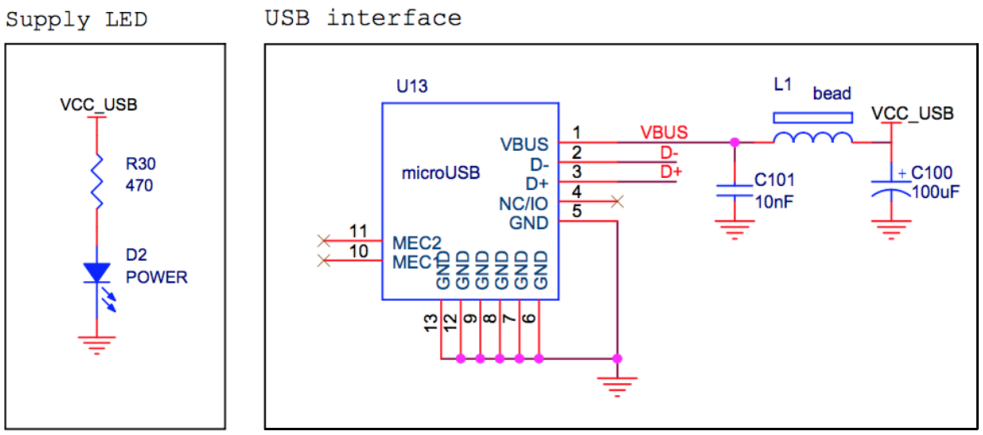


Figure 2: Schematics of mounted parts

By plugging a micro-USB into the connector, the LED should light up and a 5V power can be measured between the VCC_USB and the common GND.

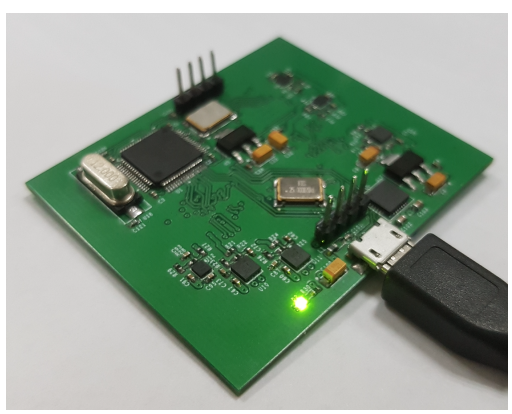


Figure 3: LED indicating proper main supply

2.3 LDOs

After this, the LDO chips with their decoupling capacitors and voltage dividers are soldered onto the PCB.

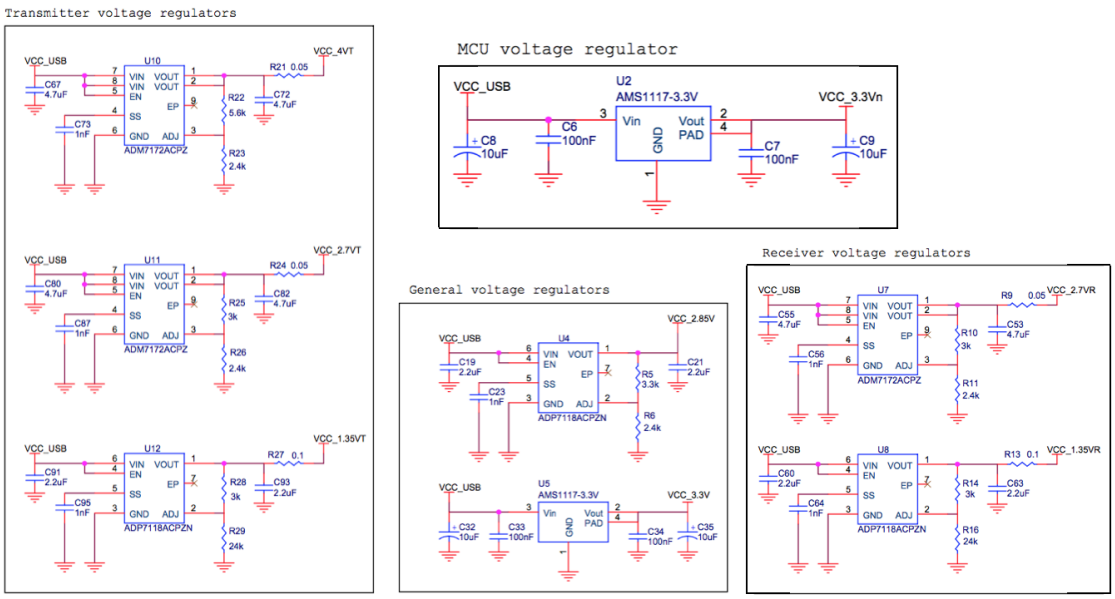


Figure 4: Schematics of mounted parts

By measuring the output of each voltage regulator with a Multimeter, the correct power supplies of the components can be confirmed.

3 SPI Simulations

For this part, the STM8 microcontroller with its capacitors and the SWIM connector are added to the board as well as the high speed external crystal of 12MHz. A software which configures the remaining programmable chips through the GPIO pins is loaded onto the MCU. The code for all the parts are located in the appendices as well. The proper functioning of this code is verified by measuring the output pins and plot the signals on an oscilloscope.

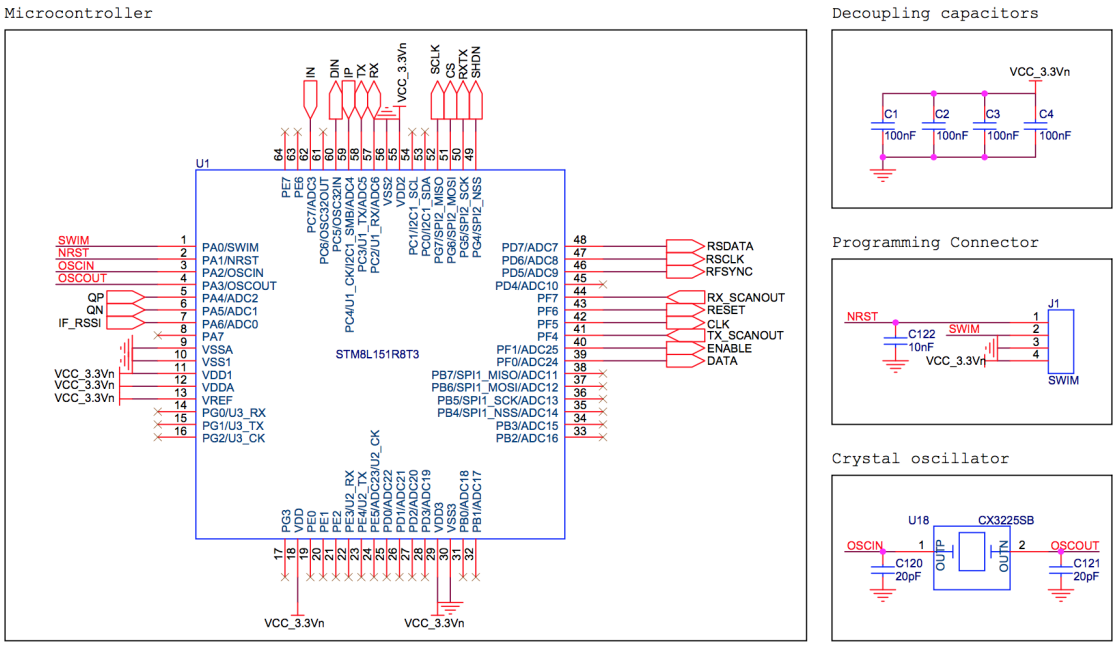


Figure 5: Schematics of mounted parts

3.1 Waveform Generator

The AD9833 has a standard 3-wire serial interface designed to load a 16-data bits word into the device. The FSYNC signal acts as a chip select bit, when this line is low, data can be transferred. On the falling edge of the SCLK line, the bit on SDATA is shifted into the register of the device. The chip select bit goes high again after 16 clock pulses. The timing diagram for this operation can be seen in Figure 6. More information about the timing parameters are described in the AD9833 datasheet.

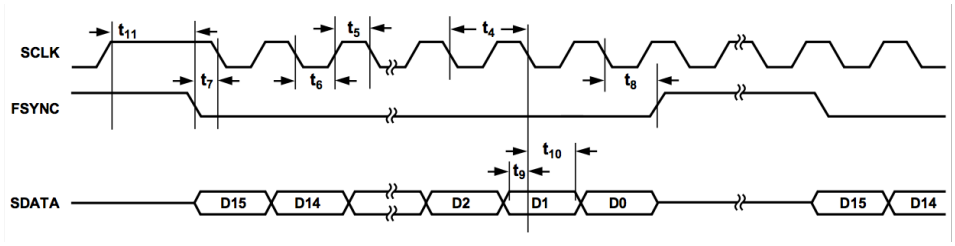


Figure 6: Serial Timing Diagram

This SPI is simulated by the microcontroller, on pins PD5 to PD7 which are configured as push-pull output ports. Figure 7 illustrates the clock signal (black) and the data signal (blue) measured on the MCU pins.

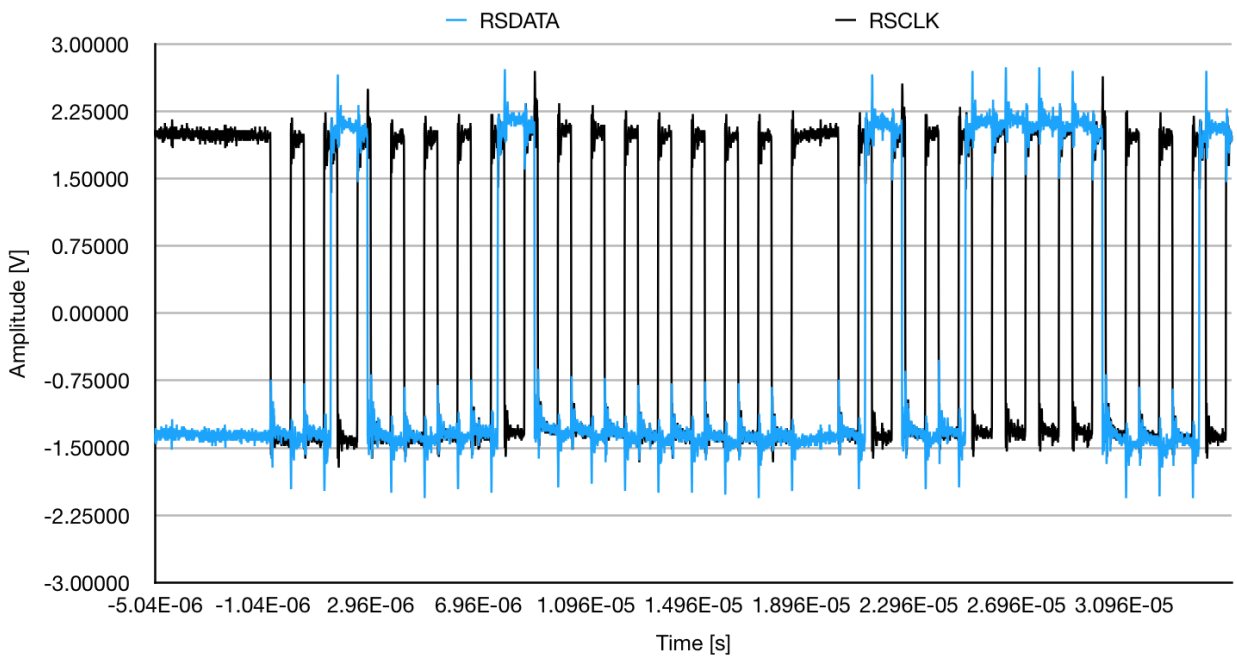


Figure 7: AD9833 SPI simulation

It can be observed that the data bit is stable on the falling edges of RSCLK and after 16 clock pulses a brief delay occurs before another write operation. As the STM8 is an 8-bit microcontroller, the code is executed at a speed that won't be faster than the given minimum periods and duration for the serial timing. The time interval for a clock high is measured at a few μs.

3.2 Transmitter & Receiver

As the transmitter HMC6300 and the receiver HMC6301 share the same sequence of signals on the SPI lines to transfer data bits and as the SPI row includes the chip address, only one serial interface is needed for both devices. Once the ENABLE line is low, data bits can be clocked in on the rising edge of CLK. A write operation requires 18 clock pulses with 18 data bits containing row data, row address, the write/read bit and the chip address. Each of these parts is clocked in LSB first. When the enable line returns high, the register array is loaded on the chips. Additionally, a reset signal is available to set the register arrays to default value. Figure 8 shows the timing diagram for this serial interface.

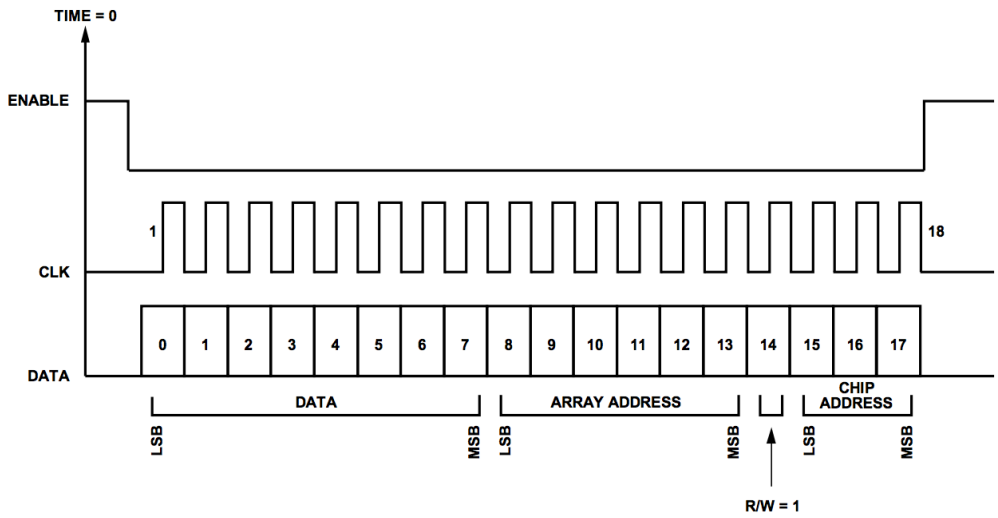


Figure 8: Timing Diagram for Writing a Row of the Transmitter Serial Interface

Simulated on the corresponding microcontroller ports of the schematics, the clock and data lines are plotted with the oscilloscope as illustrated in Figure 9 below.

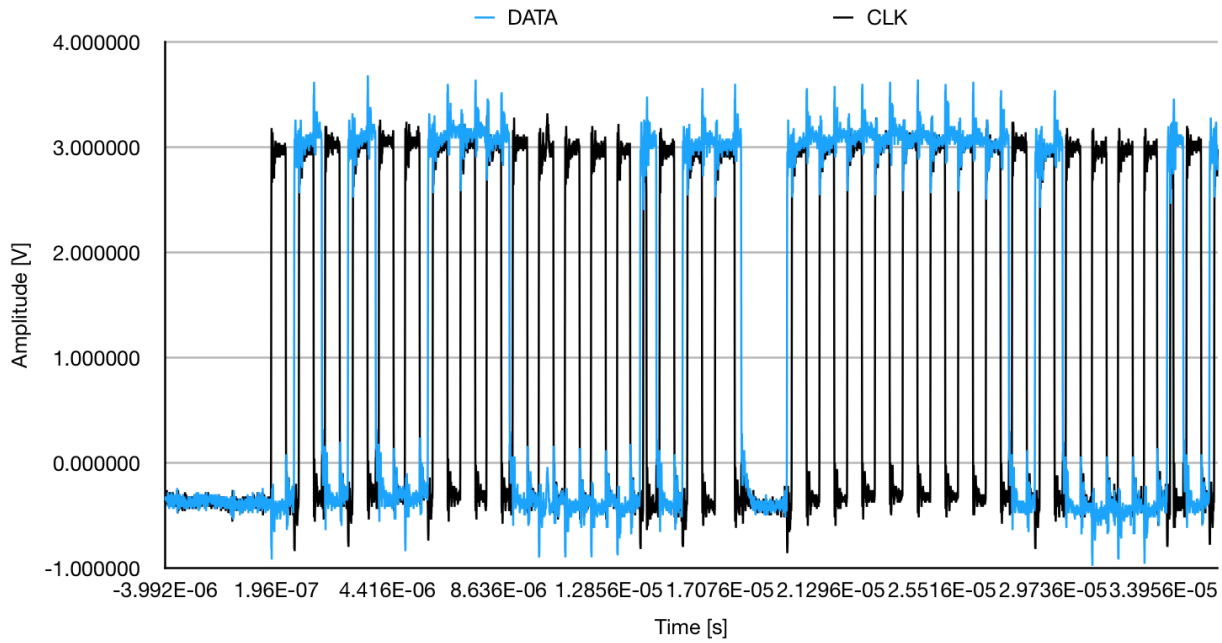


Figure 9: HMC6300 SPI simulation

This graph indicates the same principal characteristics as the timing diagram and contains 18 data bits in one row. Furthermore, by reading the last three data bits of the first row, i.e. 110 LSB first, which is the transmitter chip address, it is approved that this data transfer is destined for the HMC6300. No time limits are specified in the datasheet of these chips.

3.3 IF Downconverter

The MAX2831 includes registers with 18 data bits, programmable with a SPI interface. In this register, the first 14 bits are data and the remaining 4 indicate the register address, for both parts the MSBs are loaded first. Once the chip select bit is low, data can be shifted at the rising edge of the clock signal as shown in Figure 10. When CS returns high, the shift register is latched into the register selected by the address bits. Furthermore, logic pins SHDN and RXTX control modes of operations for this chip. More details about these modes of operations and SPI timing are given in the transceivers datasheet.

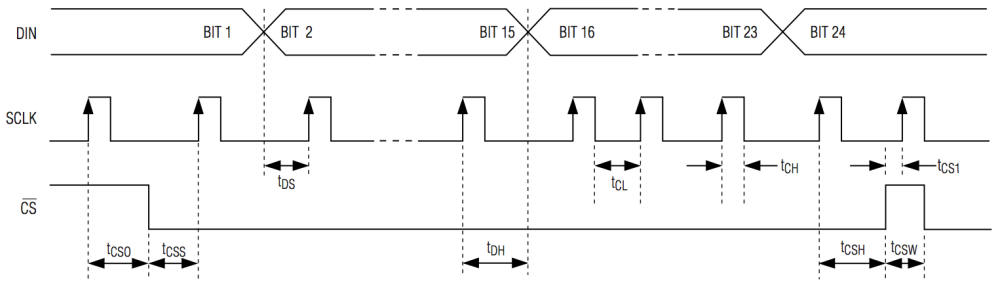


Figure 10: 3-Wire SPI Serial-Interface Timing Diagram

Pins PG4 to PG7 and PC5 are used for the SPI lines and the mode select signals. The data output, which was previously on PC0, had to be reconnected because this pin couldn't be configured as a push-pull output (see MCU datasheet). Figure 11 illustrates data signal DIN (blue) and clock signal SCLK (black) for one entire write operation.

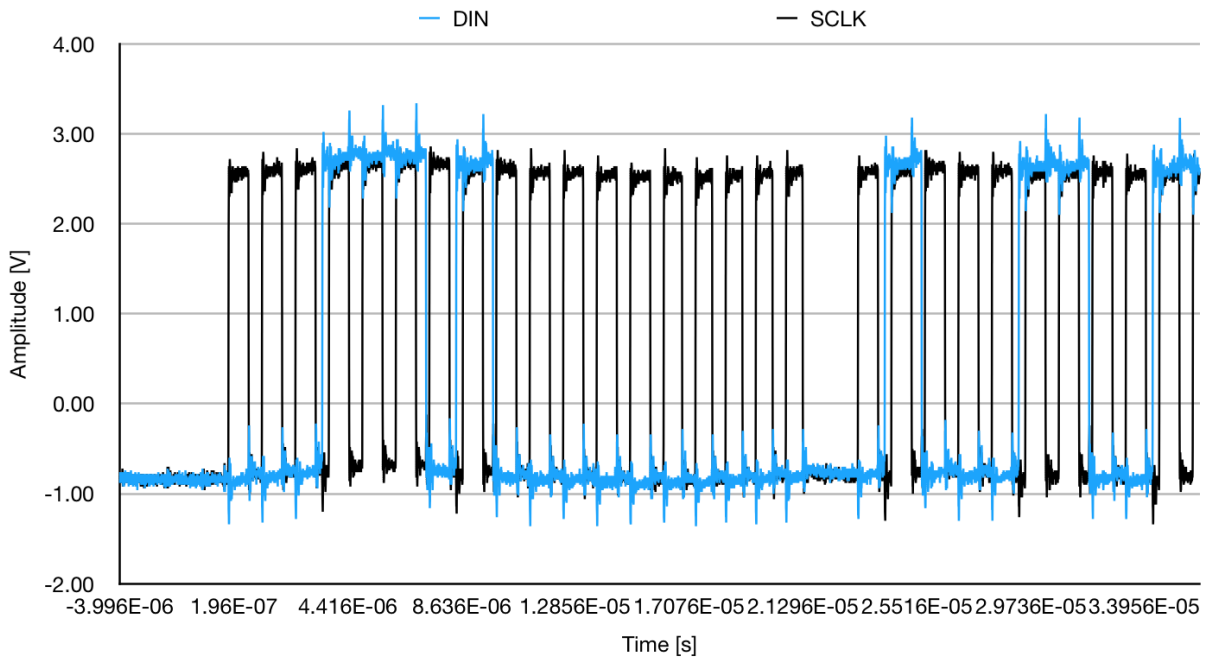


Figure 11: MAX2831 SPI simulation

By looking at the clock curve on the graph, it can be confirmed that the data line is always stable at a rising edge. Moreover, the timing characteristic of the two signals are approved by conducting several measures of the duration and periods with the oscilloscope.

4 Reference Clock generation

Next, the reference clocks for transmitter and receiver have to be generate at exactly 70MHz for these chips to work at the desired carrier frequency and to achieve an accurate down-conversion with the returning signal. For this part, the components visible in Figure 12 are soldered on the top and bottom layer of the PCB. To ensure the right operating of these devices, the frequency is measured step by step, i.e. on the outputs pins.

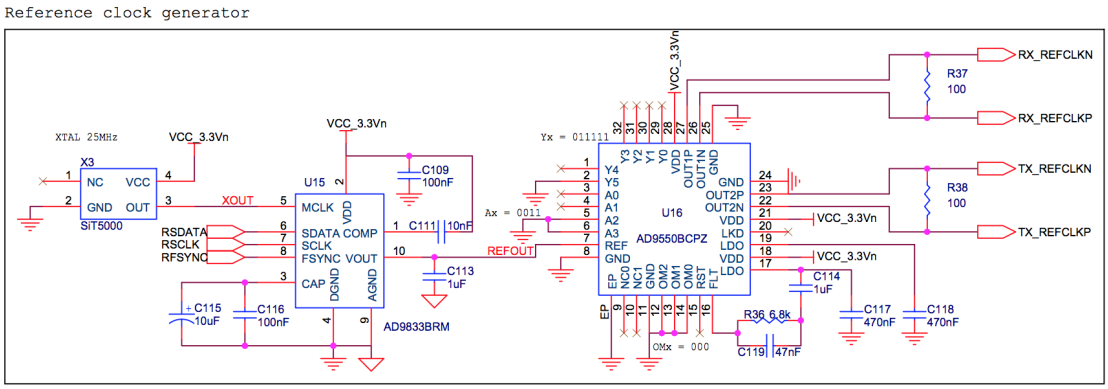


Figure 12: Schematics of mounted parts

4.1 IF clock

On the oscillator, frequencies are measured with an acquisition mode set to average 128. First, it is made sure that the crystal oscillator outputs the exact chosen frequency. Once the AD9833 is configured by the microcontroller, it should deliver 1.8436 MHz to the AD9550 as calculated previously.

Problem : When measuring on pin VOUT, no sine wave signal is observed. The reason for this error is caused by the internal output resistance and the capacitor C113 creating a low pass filter. The cutoff frequency of this filter is determined by the constant :

$$f_c = \frac{1}{2\pi R_i C_{113}} = 796Hz$$

Which completely cuts off the desired output frequency. To solve this issue, the capacitors value is reduced or simply removed from the board.

After modification, the output frequency is measured and graphed in Figure 13. This time the correct frequency can be observed.

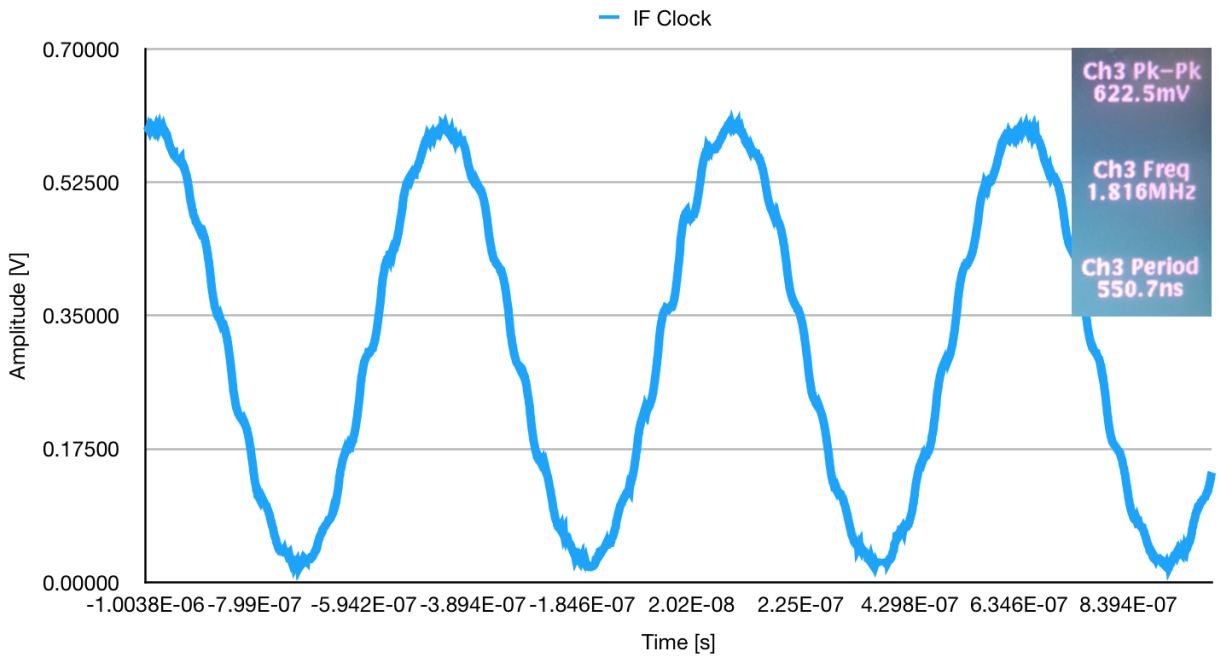


Figure 13: Intermediate frequency clock signal

4.2 Reference clock

The AD9550 accepts the intermediate reference clock as an input and translates this frequency to the required clock signal for the transmitting and receiving ICs. The translation ratio is controlled by hardwired selection pins only, thus this device is not programmed by the MCU. Two differential square wave outputs feed the HMC6300 & the HMC6301 with a reference clock frequency of 70MHz.

Problem : The measured outputs produce whether a continuous voltage high or a voltage low, i.e. 3.3V or 0V respectively. By analyzing the datasheets of the frequency translator and the waveform generator, it is noticed that the output voltage of this last chip is too low for the AD9550 input characteristics. Figure 14 shows these uncorrelated voltage ranges.

Parameter		Min	Typ	Max	Unit
AD9833	VOUT Maximum		0.65		V
	VOUT Minimum		38		mV
AD9550	Input High Voltage	1.62			V
	Input Low Voltage			0.52	V

Figure 14: AD98333 output and AD9550 input specifications

In order to make these two components compatible, a non-inverting voltage amplifier is implemented. The AD8045/8047 operational amplifier, available in the lab, is used to create a small test PCB (Figure 16) based on the schematics in Figure 15. R_S helps to reduce high frequency peaking while R_{SNUB} improves stability and minimizes ringing at the output. Additionally, two decoupling capacitors are connected between Vs and the ground. The gain A of this circuit is determined by

$$A > \frac{V_{OUT_{max}}}{V_{IN_{min}}} \Rightarrow A = 3 \tag{1}$$

Therefore, the values of the resistances can be obtained with

$$A = 1 + \frac{R_F}{R_G} \qquad R_S = R_G \qquad R_{SNUB} = R_F \tag{2}$$

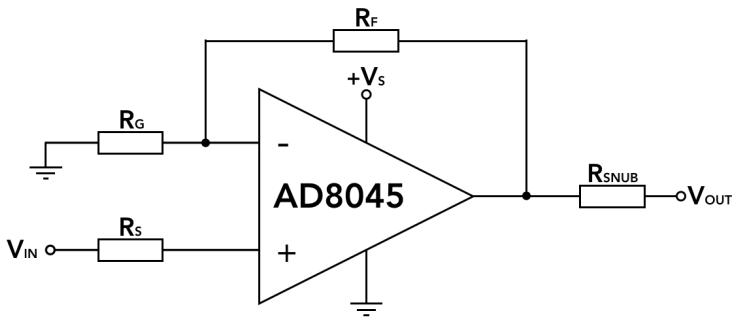


Figure 15: Non-inverting amplifier configuration

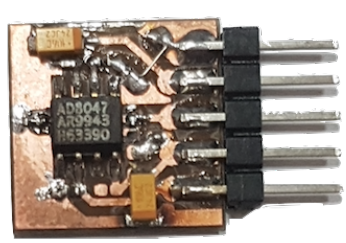


Figure 16: Non-inverting amplifier test PCB

The circuits proper functioning is confirmed by simulating it with an external frequency generator as well as power supply and by measuring the output on the oscilloscope, see Figure 17. It is then placed between the AD9833 output and the AD9550 input on the sensor board and connected to the micro USB power supply and GND.

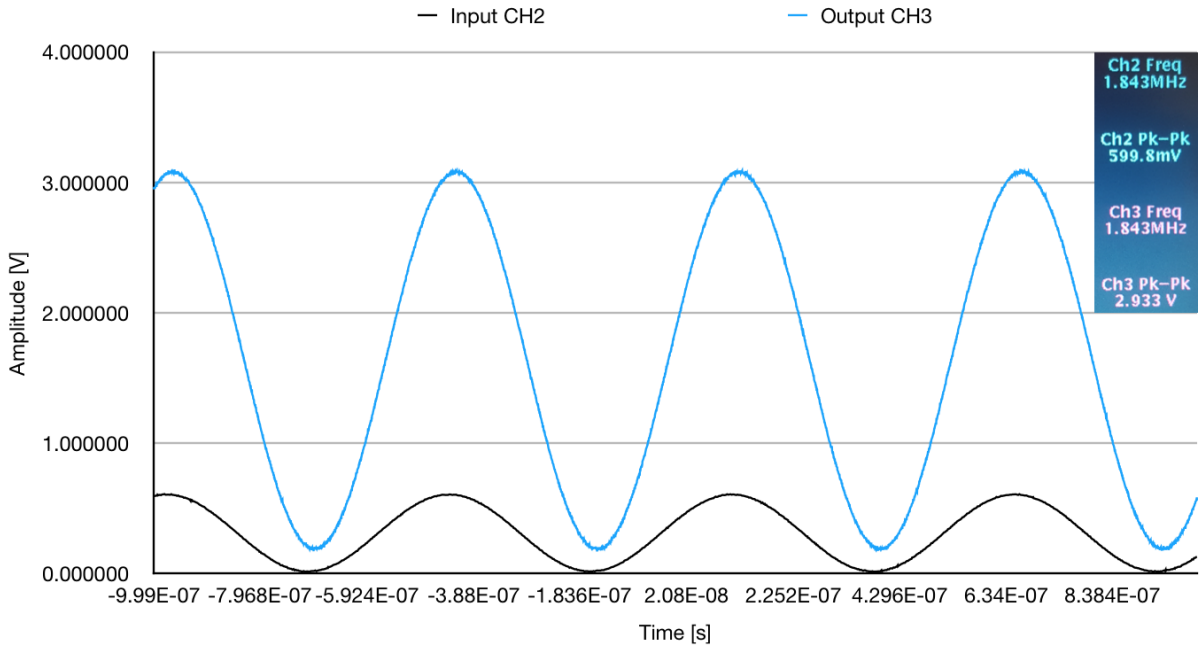


Figure 17: test PCB input & output signals

Problem : When plotting the differential signals for both chips on the outputs of the AD9550, no stable frequency can be measured. Even by using preset input and output frequencies standardized for this device, the output remains unreadable as seen in Figure 18 and 19. The two differential outputs do not show the same signals. Displayed in orange is the difference between these two measurements, as seen in the figures below it does not oscillate and remains around a static value.

For this issue no explanations and solutions could be determined before the end of the project. Another student will continue working on this project and the functioning PCB will be used for various applications including vital sign detection and drone classification.

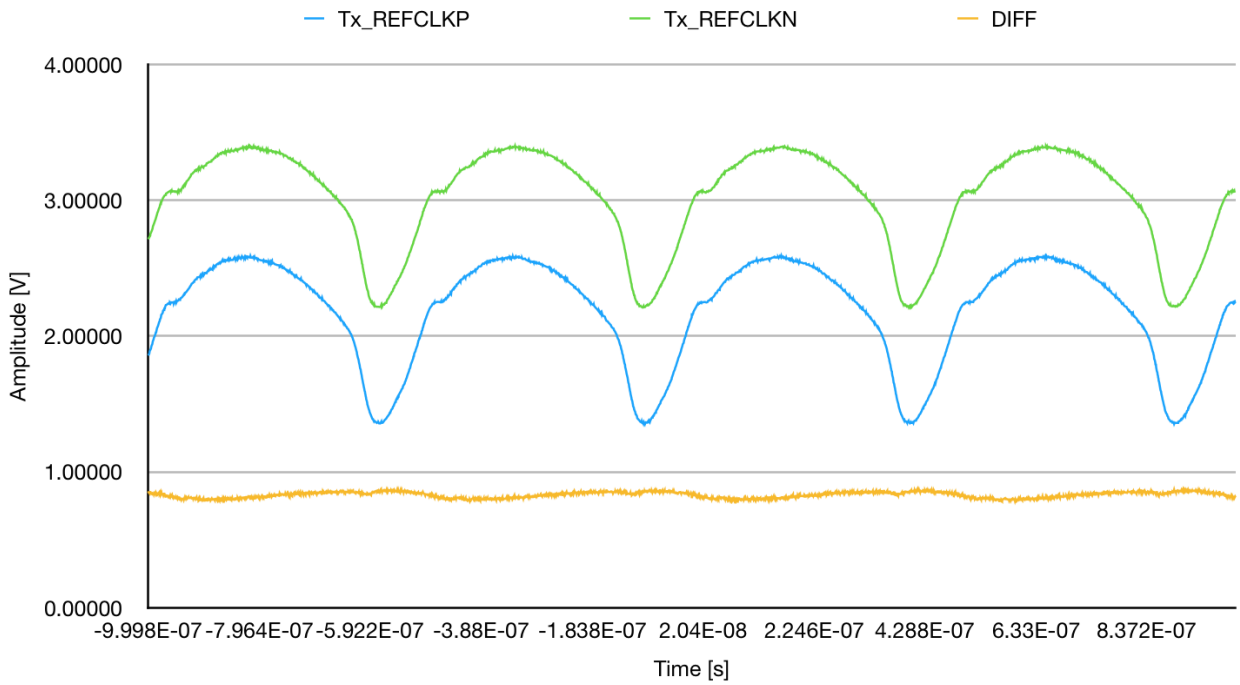


Figure 18: Differential signals and their difference for the transmitter

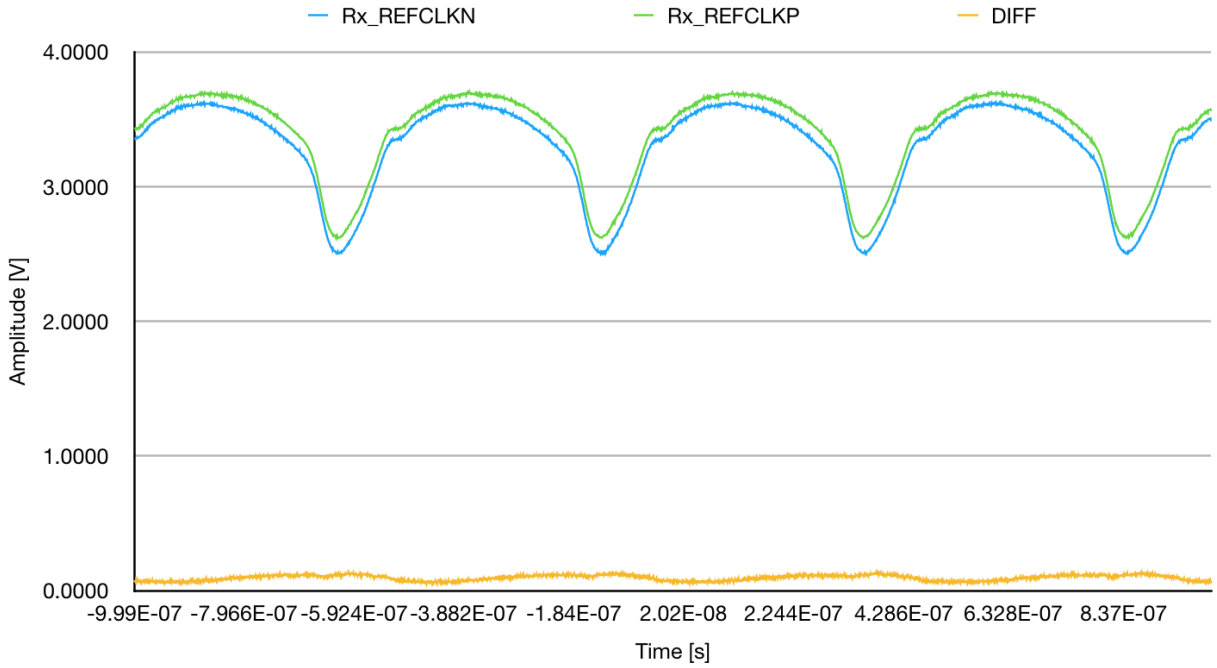


Figure 19: Differential signals and their difference for the receiver