Flying ShieldBuddy

Matthias Brandl (IFAT PMM TI COP)

Introduction

The Aurix ShieldBuddy is an Arduino Shield with an Arduino MEGA form factor and contains an Aurix TC275C. It can be programmed using the Arduino language or in C with an Eclipse IDE. A guide for the full installation of the toolchain required to program the ShieldBuddy with Infineon Low Level Drivers (ILLDs) can be found on the myICP site "Drones" under the subfolders "Drones \ FlyingShieldBuddy \ Documentation".
In combination with the FlyingShield, the ShieldBuddy can be used as a flight controller of a quadcopter. This document describes the hardware interface to the FlyingShield and the software on the Aurix. The actual flight controller is not implemented. The idea of this software is to provide a platform for a flight controller. The controller itself can then be implemented in Matlab Simulink and embedded in the software framework with Matlab's automatic code generation.

Hardware Interface to the FlyingShield

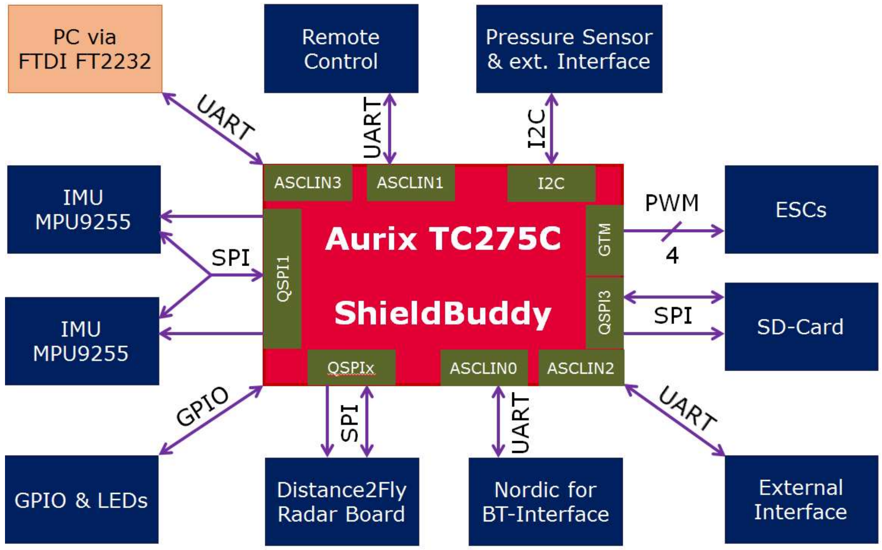

Figure 1: Block diagram of the ShieldBuddy and its connections to the peripherals.

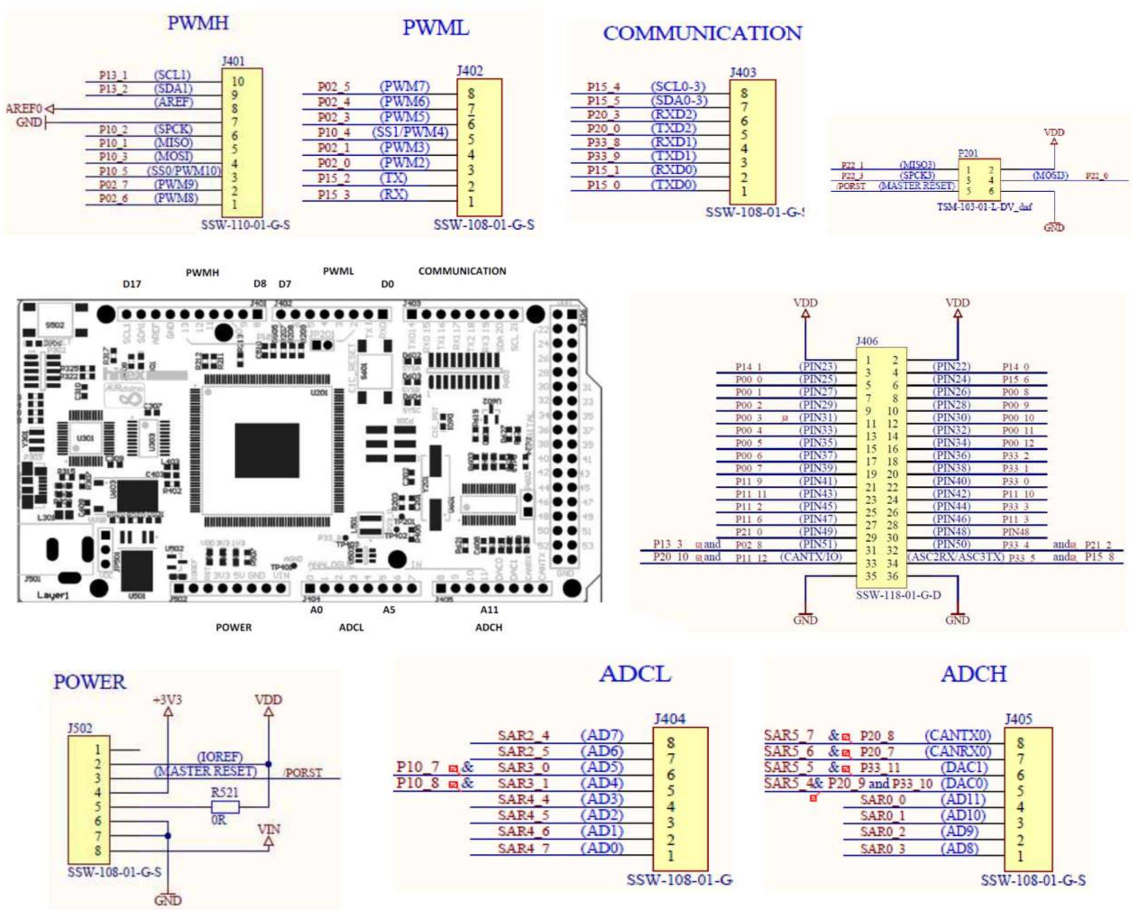

Figure 2: The ShieldBuddy layout and pinout

Table 1: Pinout of the ShieldBuddy

Aurix-Pin	Module	Function	ShieldBuddy-Pin
P2.0	SCU	Ext. Interrupt from IMU 1	PWML.3
P2.1	QSPI3.Master - CS	CS for SD-card	PWML.4
P2.3	GTM-TOM0	PWM for ESC1	PWML.6
P2.4	GTM-TOM0	PWM for ESC2	PWML.7
P2.5	GTM-TOM0	PWM for ESC3	PWML.8
P2.6	QSPI3.Master - MOSI	SD card	PWMH.1
P2.7	GTM-TOM0	PWM for ESC4	PWMH.3
P10.1	QSPI1.Master - MISO	IMU readout	PWMH.5
P10.2	QSPI1.Master - clock	IMU readout	PWMH.6
P10.3	QSPI1.Master - MOSI	IMU readout	PWMH.4
P10.4	QSPI1.Master - CS	IMU readout - IMU 2	PWML.5
P10.5	QSPI1.Master - CS	IMU readout - IMU 1	PWMH.3
P10.7	QSPI3.Master - MISO	SD card	ADCL.5(PIN 6)
P10.8	QSPI3.Master - clock	SD card	ADCL.6(PIN 5)
P11.2	GPIO	Debug	XIO.26 Do not connect
P11.6	GPIO	Debug	XIO.28 Do not connect
P13.1	I2C.SCL	Pressure Sensor & ext. Interface	PWMH.10
P13.2	I2C.SDA	Pressure Sensor & ext. Interface	PWMH.9
P15.1	ASCLIN1.UART - RX	Remote Control	COMMUNICATION.7
P15.2	ASCLIN0.UART - TX	Nordic - BT Interface	PWML.2
P15.3	ASCLIN0.UART - RX	Nordic - BT Interface	PWML.1
P15.4	QSPI2.Master - MISO	Distance2Fly Radar Board	COMMUNICATION.2
P15.5	QSPI2.Master - MOSI	Distance2Fly Radar Board	COMMUNICATION.1
P15.6	QSPI2.Master - clock	Distance2Fly Radar Board	XIO.5
P15.7	ASCLIN3.UART - TX	PC	Not on Shield
P20.0	SCU	Ext. Interrupt from IMU 2	COMMUNICATION.4
P20.3	QSPI2.Master - CS	Distance2Fly Radar Board	COMMUNICATION.3
P20.10	QSPI2.Master - CS	Distance2Fly Radar Board	XIO.34
P21.0	GTM-TOM0	Inverse PWM	XIO.30 Do not connect
P21.1	GTM-TOM0	Inverse PWM	XIO.29 Do not connect
P21.2	GTM-TOM0	Inverse PWM	Do not connect
P33.2	ASCLIN3.UART - RX	PC	Not on Shield
P33.8	ASCLIN2.UART - RX	External Interface	COMMUNICATION.5
P33.9	ASCLIN2.UART - TX	External Interface	COMMUNICATION.6
P14.1	CCU60 output PWM	IMU SYNC (FSYNC)	XIO.4

P00.9	GPIO	LED green	XIO.9
P00.10	GPIO	LED blue	XIO.11
P00.11	GPIO	LED red	XIO.13

The logic level of the digital pins is 5V.

Software overview

The Aurix TC275C on the ShieldBuddy has three cores. The first core (CPU0) is responsible for the handling the sensor data input and actuator output. Hence, it reads out the inertia measurement unit (IMU) and receives data from the remote control (RC). Once all data required for the flight controller algorithm is acquired, the flight controller algorithm is executed in the second core (CPU1). The flight controller sets the values for the electronic speed controls and CPU0 updates the values on the next interrupt.
The standard software framework does not include the flight controller algorithm. The ShieldBuddy is considered as a part of an educational quadcopter. The software framework should provide the sensor data to the flight controller algorithm. The algorithm itself needs to be manually programmed by students, either in ansi-C or in Matlab Simulink with automatic code generation.

Configuration

The Aurix contains a flash memory in which some information for the flight controller is stored. This information is

- The name of the drone (maximum 31 characters).
- The type of the remote control to know what communication to use.
- The IMU type to be able to work with different IMUs.
- The IMU accelerometer full scale range for the IMU configuration.
- The IMU gyro full scale range for the IMU configuration.
- The IMU average length (as a simple low pass filter of the current values). Possible values are powers of 2 between 1 and 64.
- The type of PWM generation – standard is a PWM signal from the GTM for each ESC.
- Mapping of the physical ESCs to the ESCs in the algorithm, e.g. ESC1 is +X and –Y.
- A number that defines how often the flight controller algorithm is executed. It can range from every time the IMU was read to only every 128th time the IMU was read (in powers of 2).

Inertia Measurement Unit (IMU)

So far, only one type of IMU has been implemented, MPU9250. For fast readout, the IMU is read via an SPI-interface instead of the slow I2C-interface. The sensor that was tested was a MPU9255. According to the datasheet, maximum SPI frequency is 1 MHz. But during the tests, SPI communication with 10 MHz was successful which allows a full readout of the accelerometer and the gyro sensor even at the highest sampling frequency of 32 kHz.
During initialization, the sampling of the gyro sensor is set to 32 kHz (maximum) and the sampling of the accelerometer is set to 4 kHz (maximum). If the sensor data has to be filtered, this can be performed in the microcontroller rather than in the IMU, where it is not perfectly clear what the filter really does.
In the configuration, one can define the full scale range of the accelerometer and the gyro sensor as well as an averaging length. This averaging length is always a power of 2 and it defines over how many samples the sensor data is averaged. The output of the IMU is always a 16 bit number with highest full scale range ($\pm$16g or 2000°/s) independent of the actual full scale range or the averaging length.

Electronic Speed Control (ESC)

So far, only one type of communication with the ESCs has been implemented. The Generic Timer Module (GTM) generates 4 PWM signals that control the 4 motors of the quadcopter. The PWM frequency is 480 Hz. After each PWM cycle, an interrupt service routine is executed which sets the PWM duty cycle.

Remote Control (RC)

not yet implemented – S-bus is inverted UART and I have not yet found a way to invert UART in the Aurix. (maybe hardware inverter??)

USB- and Bluetooth-connection to PC

In order to flash the configuration, as described in Section Configuration, into the memory of the Aurix, one can use either the micro-USB connector on the ShieldBuddy via UART or an UART connection to a chip with Bluetooth on the Shield. Both UART communications have a baud rate of 115200 baud/s, 1 stop bit and even parity.
The communication is with strings of which the first part is the command. The command is followed by '?' for a get command and '=' for a set command. A set command has a value string attached to whole string. The result of a get command is a string with the desired variable. The result of a set command is an acknowledge (char 06). Unknown command do not provoke a response.

Command	Possible Values	Description
NAME		31 character long name
AURIX	TC275C	Only get command – like a "who are you?"
RC	SBus Unknown	Type of remote control
IMU	MPU9250 Unknown	Type of IMU
ACC_FS	2g 4g 8g 16g	Full scale range of the accelerometer
GYRO_FS	250dps 500dps 1000dps 2000dps	Full scale range of the gyro sensor
IMU_AVG	1 2 4 8 16 32 64	IMU average length
PWM	GTM Unknown	PWM generation type
ESC_+X_+Y	ESC1 ESC2 ESC3 ESC4	ESC number of the ESC +X and +Y of the algorithm
ESC_-X_+Y	ESC1 ESC2 ESC3 ESC4	ESC number of the ESC -X and +Y of the algorithm
ESC_+X_-Y	ESC1 ESC2 ESC3 ESC4	ESC number of the ESC +X and -Y of the algorithm
ESC_-X_-Y	ESC1 ESC2 ESC3	ESC number of the ESC -X and -Y of the algorithm

	ESC4	
FC_ALGO_EXE	250Hz 500Hz 1kHz 2kHz 4kHz 8kHz 16kHz 32kHz	Flight controller algorithm execution frequency
SAVE		Saves the configuration to the flash memory.
INIT		Initializes the configuration to standard values.
RESET		Resets the device.

Data logging

not yet implemented

Radar-Interface

not yet implemented