ROCKETuC Protocol

2012
Alexander Reben
Stefan Wendler

Input Packets
- General Packet format
- Packet type
  - Pin function (setup)
  - Pin control
  - PWM function (setup)
  - PWM control
- CRC

Return Packets
- General Packet format
- Packet type
  - System info
    - Hardware type
  - STATUS / ERROR
  - Digital pin read
  - Analog pin read
- CRC

Example Packages

Input Packets

General Packet format

Byte #	1	2	3	n	4+n
Description	Start of packet 0x24 $	Packet length (total bytes including predefined)	Packet type	Data (Defined by packet type)	CRC

Packet type

Description	Value	Data length (bytes)
NULL	0x00	0
RESERVED	0x01	n
System Info	0x02	2
Device control (reserved for later)	0x03	n
Pin function (setup)	0x04	2
Pin control	0x05	2
PWM function (setup)	0x06	3
PWM control	0x07	3
Serial function (setup)	0x08	?
Serial data	0x09	n
External Interrupt function (setup)	0x0A	2
RESET	0xFF	0

Pin function (setup)

Byte	Description
1	Pin number (0x PORT PIN, i.e. P2.3 -> 0x23)
2	Pin function

Description	Byte 2 value
Set pin as input float	0x00
Set pin as input pull-up	0x01
Set pin as input pull-down	0x02
Set pin as output	0x03
Set pin as analog in	0x04
Set pin as PWM	0x05
Set pin as serial TX	0x06
Set pin as serial RX	0x07

Pin control

Byte	Description
1	Pin number
2	Pin value

Description	Byte 2 value
Clear pin	0x00
Set pin	0x01
Toggle pin	0x02
Digital pin read	0x03
Analog pin read	0x04

Pulse length read (PWM read)	0x05

PWM function (setup)

Byte	Description
1	Pin number
2	PWM period LSB
3	PWM period MSB

Description	Byte 2+3 value
PWM period in ms	0x0000 - 0xFFFF

PWM control

Byte	Description
1	Pin number
2	PWM duty cycle

Description	Byte 2 value
PWM duty cycle	0x00 - 0xFF 0x00 = 0%, 0xFF = 100%

CRC

Simple CRC is calculated by adding up bytes 2 to n (all bytes btw. package start and CRC). In C code this would look something like this:

```
unsigned char *pkt;         // pointer to whole package
unsigned char  crc = 0;     // resulting 1 byte CRC

for(int i = 1; i < pkt[1] - 1; i++) crc += pkt[i];
```

Return Packets

General Packet format

Byte #	1	2	3	n	4+n
Description	Start of packet 0x2B +	Packet length (total bytes including predefined)	Packet type	Data (Defined by packet type)	CRC

Packet type

Description	Value	Data length (bytes)
NULL	0x00	0
STATUS / ERROR	0x01	1
System Info	0x02	2
Digital pin read	0x03	2
Analog pin read	0x04	3
Pulse length read	0x05	3
RESET	0xFF	0

System info

System parameter	Byte 1	Byte 2
No info	0x00	0x00
Hardware type	0x01	n
Firmware rev	0x02	n

SW: I think I don't get this …, is for each system parameter a separate message sent?

I also would suggest to have “Hardware typ” separated into “Board type” and “MCU type”, since potentially I could use different MCUs in the Launchpad, or have a MSP430G2553 in something else than a Launchpad …

The problem here is how is the MCU going to know which board it is on?
This was intended to be a check that returns what firmware for what device is loaded.
What improvements do you think can be made?

Hmm that's pretty true … it does not know unless every target has its own firmware binary (which should be avoided to keep variants to maintain low). I remember a project where they were dealing with different peripheral hardware on the same MCU (which is similar to our board situation). They solved the problem by storing this additional information in the internal flash of the MCU. The problem here is, that again the information has to get to the flash. What they did in this project I mentioned was to put that information at first firmware flash on the device too (there was a custom made bootloader supporting this). Maybe that's overkill here for a first shot.

Pragmatically we could start with returning MCU type (which is known to the firmware), and don’t return board type at all …, or better way, return board type, and start with “Launchpad” hardcoded …

What I still don’t get is, what do you mean with “Bit 1”, “Bit 2” in the header, is this Byte 1 / 2 ?

Perhaps it would be better to do what you suggested and have the MCU report its capabilities.
That way, each firmware for each specific chip would be written to know what it can handle.
I assume that firmware for an MCU on the launchpad would be different from a bare MCU.
So for example there could be a MSP430 2335 on launchpad firmware and a bare MSP430 2330 firmware.
I think this can wait to be implemented later though.

Hardware type

Description	Byte 2 value
Unknown	0x00
Launchpad	0x01

STATUS / ERROR

Status type	Value
Unknown	0x00

ACK	0x01
Bad CRC / Malformed packet	0x02
Invalid packet type	0x03
Invalid data	0x04
Invalid pin command	0x05

Digital pin read

Byte	Description
1	Pin number
2	Pin value

Description	Byte 2 value
Pin low	0x00
Pin high	0x01

Analog pin read

Byte	Description
1	Pin number
2	Analog value LSB
3	Analog value MSB

CRC

see CRC for *Input Packages*

Example Packages

Read System Info

Send system info request packet to MCU

0x24	0x04	0x02		0x06
Start of packet *$*	Packet length *4 Bytes*	Packet type *System Info*	Data *Empty*	CRC

Result packet received from MCU **on Success**

0x2B	0x07	0x02	0x01 0x01 0x01	0x0C
Start of packet +	Packet length *7 Bytes*	Packet type *System Info*	Data *[0] Board-Type* *[1] MCU-Type* *[2] Firmware Rev.*	CRC

Digital output

Configure pin as output

Send pin function output to MCU

0x24	0x06	0x04	0x14 0x03	0x21
Start of packet *$*	Packet length *6 Bytes*	Packet type *Pin function (setup)*	Data *[0] Pin P1.4* *[1] Set pin as output*	CRC

Result received from MCU **on Success**

0x2B	0x05	0x01	0x01	0x07
Start of packet +	Packet length *5 Bytes*	Packet type *STATUS / ERROR*	Data *[0] ACK*	CRC

Set output pin to HIGH

Send pin control HIGH to MCU

0x24	0x06	0x05	0x14 0x01	0x20
Start of packet *$*	Packet length *6 Bytes*	Packet type *Pin control*	Data *[0] Pin P1.4* *[1] Set to HIGH*	CRC

Result received from MCU **on Success**

0x2B	0x05	0x01	0x01	0x07
Start of packet +	Packet length *5 Bytes*	Packet type *STATUS / ERROR*	Data *[0] ACK*	CRC

Digital input

Configure pin as input with pull-down enabled

Send pin function input pull-down to MCU

0x24	0x06	0x04	0x15 0x02	0x21
Start of packet *$*	Packet length *6 Bytes*	Packet type *Pin function (setup)*	Data *[0] Pin P1.5* *[1] Set pin input with pull-down enabled*	CRC

Result received from MCU **on Success**

0x2B	0x05	0x01	0x01	0x07
Start of packet +	Packet length *5 Bytes*	Packet type *STATUS / ERROR*	Data *[0] ACK*	CRC

Read input state

Send pin control read to MCU

0x24	0x06	0x05	0x15 0x03	0x23
Start of packet *$*	Packet length *6 Bytes*	Packet type *Pin control*	Data *[0] Pin P1.5* *[1] Digital pin read*	CRC

Result received from MCU **on Success**

0x2B	0x06	0x03	0x15 0x01	0x1F
Start of packet +	Packet length *6 Bytes*	Packet type *Digital pin read*	Data *[0] Pin P1.5* *[1] Pin state is HIGH*	CRC

Analog input

Configure pin as analog input

Send pin function analog input to MCU

0x24	0x06	0x04	0x20 0x04	0x2E
Start of packet *$*	Packet length *6 Bytes*	Packet type *Pin function (setup)*	Data *[0] Pin P2.0* *[1] Set pin analog input*	CRC

Result received from MCU **on Success**

0x2B	0x05	0x01	0x01	0x07
Start of packet +	Packet length *5 Bytes*	Packet type *STATUS / ERROR*	Data *[0] ACK*	CRC

Read input state

Send pin control read to MCU

0x24	0x06	0x05	0x20 0x04	0x2F
Start of packet *$*	Packet length *6 Bytes*	Packet type *Pin control*	Data *[0] Pin P2.0* *[1] Analog pin read*	CRC

Result received from MCU **on Success**

0x2B	0x07	0x04	0x20 0x00 0xAB	0xD6
Start of packet +	Packet length *7 Bytes*	Packet type *Analog pin read*	Data *[0] Pin P2.0* *[1] LSB of ADC sample* *[2] MSB of ADC sample*	CRC

Using PWM

Configure pin as PWM

Send pin function PWM to MCU

0x24	0x06	0x04	0x21 0x05	0x30

Start of packet	Packet length	Packet type	Data	CRC
$	*6 Bytes*	*Pin function (setup)*	*[0] Pin P2.1* *[1] Set pin as output*	

Result received from MCU **on Success**

0x2B	0x05	0x01	0x01	0x07
Start of packet	Packet length	Packet type	Data	CRC
+	*5 Bytes*	*STATUS / ERROR*	*[0] ACK*	

Setup PWM

Send PWM function (setup) to MCU

0x24	0x07	0x06	0x21 0x00 0x14	0x42
Start of packet	Packet length	Packet type	Data	CRC
$	*7 Bytes*	*PWM function (setup)*	*[0] Pin P2.1* *[1] Period in ms LSB* *[2] Period in ms MSB (20ms)*	

Result received from MCU **on Success**

0x2B	0x05	0x01	0x01	0x07
Start of packet	Packet length	Packet type	Data	CRC
+	*5 Bytes*	*STATUS / ERROR*	*[0] ACK*	

Change duty cycle

Send PWM control to MCU

0x24	0x06	0x07	0x21 0x13	0x41
Start of packet	Packet length	Packet type	Data	CRC
$	*6 Bytes*	*PWM control*	*[0] Pin P2.1* *[1] duty cycle (7.5% ~ 1.5ms)**	

*) Period is set to 20ms
1.5ms is 7.5% of 20ms
100% duty cycle equals 0xFF (255)
Thus, 1% equals 2.55, and 7.5% equals 19.125 (~0x13)

Result received from MCU **on Success**

0x2B	0x05	0x01	0x01	0x07
Start of packet +	Packet length *5 Bytes*	Packet type *STATUS / ERROR*	Data *[0] ACK*	CRC