CG1112 AY17/18 quiz 2 cheatsheet compiled by linruo@comp.nus.edu.sg

USART Communications

- Asynchronous mode used for EPP2 – the transmission is independent of clock signal.
- The transmitter uses an internal clock to determine when to send each bit.

1. Physical Layer

- RX: receives incoming bits (0)
- TX: transmits outgoing bits (1)

2. Data Link Layer

- Both sides must agree on the number of data bits, type of parity bits, number of stop bits and same bit rate / baud rate.

3. Registers

- UCSR0C (pair up with UCSR0B to set the 3rd bit of the data size)

UMSEL01	UMSEL00	UPM01	UPM00	USBS0	UCSZ01	UCSZ00	UCPOL0
							0

UMSEL0[1:0]	Mode
00	Asynchronous USART
01	Synchronous USART
10	Reserved
11	Master SPI (MSPIM)[1]

UPM0[1:0]	ParityMode
00	Disabled
01	Reserved
10	Enabled, Even Parity
11	Enabled, Odd Parity

USBS0	Stop Bit(s)
0	1-bit
1	2-bit

UCSZ0[2:0]	Character Size
000	5-bit
001	6-bit
010	7-bit
011	8-bit
100	Reserved
101	Reserved
110	Reserved
111	9-bit

- $B = \frac{f_{osc}}{16\times baud} - 1$ to set the lower byte of UBBR0L If B < 255
- UBBR0H = 0 (for input remaining bits)
- UBBR0H = input remaining bits if B > 8 bits

-UCSR0B

RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0	UCSZ02	RXB80	TXB80

Bit	Label	Comment
7	RXCIE0	Set to 1 to trigger USART_RX_vect interrupt when a character is RECEIVED.
6	TXCIE0	Set to 1 to trigger USART_TX_vect interrupt when finish sending a character.
5	UDRIE0	Set to 1 to trigger USART_UDRE_vect interrupt when sending data register is empty.
4	RXEN0	Enable USART receiver.
3	TXEN0	Enable USART transmitter.
2	UCSZ02	Used with UCSZ01 and UCSZ00 (bits 2 and 1) bits in UCSR0C to specify data size.
1	RXB0	9th bit received when operating in 9-bit word size mode.
0	TXB0	9th bit to be transmitted when operating in 9-bit word size mode.

-UCSR0A (contain flag that tells us whether we can send or read data)

RXC0	TXC0	UDRE0	FE0	DOR0	UPE0	U2X0	MPCM0

7	RXC0	Becomes 1 when data is received
6	RXC0	Finished sending data
5	UDRE0	Data register is empty
4	FE0	Become 1 if frame error
3	FOR0	Data overrun
2	UPE0	Parity error
1	U2X0	
0	MPCM0	

ADC Programming

- Nyquist Sampling Rate – sampled signal frequency should be at least twice of the highest frequency of the signal
- Input voltage signals are mapped in the range of 0 -5V (Quick math)
- Resolution: voltage distance between two adjacent quantization levels. Higher quantization levels, higher accuracy.
- N bits data divide the input signal range into 2^b diff quantization levels.

$$resolution = (voltage\ span)/2^b = (V_{ref\ high} - V_{ref\ low})/2^b$$

Power Reduction Register (turn off power to save energy)

PRTWI0	PRTIM2	PRTIM0		PRTIM1	PRSPI0	PRUSART0	PR-ADC

0	PR-ADC	Write 0 to activate the power of ADC

ADCSRA

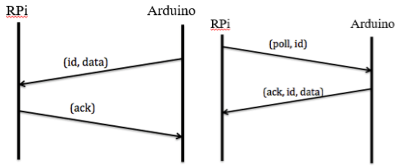
ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

7	ADEN	Write 1 to switch on ADC
6	ADSC	Write 1 t start the conversation / Poll it until it becomes 0

- $f_s = \frac{f_{clk}}{ps}$ to find the scalar value from the sampling frequency

CG1112 AY17/18 quiz 2 cheatsheet compiled by linruo@comp.nus.edu.sg

Communication Protocols



- Left: periodic push by Arduino -> Rpi needs to buffer the incoming data
- Right: periodic poll by RPi -> potential loss of data on arduino if not enough polling

Finding Checksums

-To check if data is received correctly.

- b1 XOR b2 XOR b3 XOR …

Serializing Structures

-Convert a packet structure of data into stream of bytes

-Deserialise the stream of bytes back to strutures

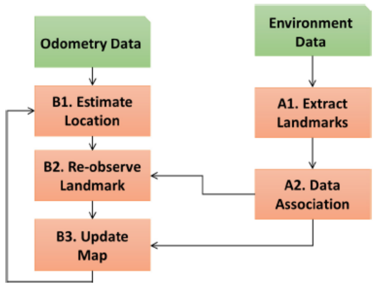
1. Endianness

- Big endian: higher order bytes at lower addresses
- Little Endian: lower order bytes at lower addresses
- Need to convert all data to standard endianness
- Convert it back to the native endianness at the destination

2. Differing Data Types

- Arduino: int is 16 bits
- Pi: 32 bits and receive 4 bytes by 4 bytes of data
- need to convert them using int32_t
- use padding to makesure that the packet sent by Arduino and received at Pi are equal
- Magic number to ensure it's a valid packet

Simultaneous Localization and Mapping



Hector – doesn't require odometry data but cant handle big movements between updat

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

ADMUX (Choose and convert reference voltage)

REFS1	REFS0	ADLAR		MUX3	MUX2	MUX1	MUX0

REFS[1:0]	Voltage Reference Selection
00	AREF, Internal V_{ref} turned off
01	AV_{CC} with external capacitor at AREF pin
10	Reserved
11	Internal 1.1V Voltage Reference with external capacitor at AREF pin

Channel	MUX2	MUX1	MUX0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Transfer Control Protocol / Internet Protocol

-Defines how many computers/clients can set up communications over a network to reach the server at the specific IP address

- IP/Network layer protocol': deals how data is transmitted from one end to another through a network of routers and hosts. Routers often drop packets rather than delivering them when its overwhelmed with data and its buffers are full. Thus IP doesn't guarantee actual packet delivery.
- TCP solves this problem by establishing a connection between source and destination. It reassembles the data in the correct order.
- TCP/IP uses a special data structure – socket to open connection to a host and read/write data to host

Transport Layer Security

1. Alice writes a message M and hash it into D
2. Encrypt D to get D' with Alice's private key and use Bob's public key to encrypt (M+D') to form C
3. Bob decrypt C with his own private key to get (M+D') and hash it to get D
4. Bob uses Alice's public key to decrypt D' (aka digital signature)

Certificates

1. Alice gives Charlie her public key and Charlie derives a hash for her and encrypt it with his private key
2. Bob computes the hash and use Charlie's public key to decrypt the signature.