Lab 7: Controller Area Network

Lab 7: Controller Area Network

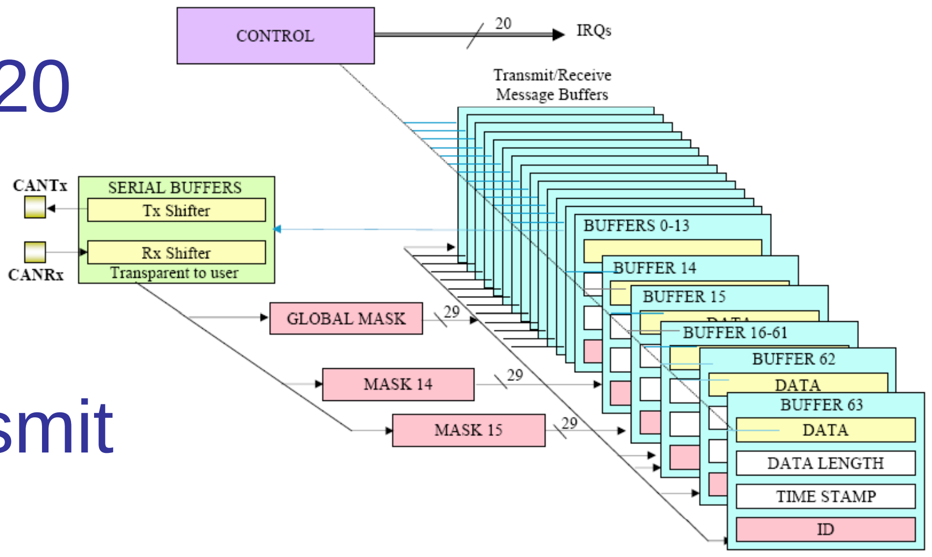
- For a general description of CAN, see the document posted on the course website
- Lab 7 has two parts:
  - Part 1: you will write a program that communicates with a neighboring lab station to implement the virtual spring-wall on adjacent haptic wheels
  - Part 2 “virtual chain”: Each haptic wheel is connected to adjacent wheels with virtual springs and dampers. When all groups are online, moving your wheel will push and pull on the wheels of your immediate neighbors, and if no one is holding the other wheels, all the wheels in the lab can be moved from just one lab station.

MPC5553 FlexCAN

- 3 FlexCAN2 Modules
  - We will use FlexCAN “A”
- Up to 64 message buffers each
  - Holds queue of received messages and messages ready to be transmitted
- Maskable interrupt
  - Jump to ISR when message is received
- Debugging mode available
  - Programmable loop-back for self test operation
- See Chapter 22 of Reference Manual

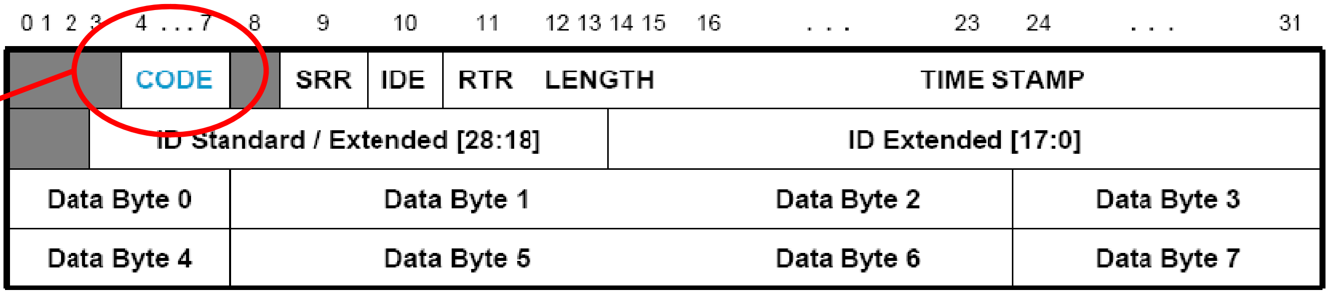
MPC5553 FlexCAN Message Buffers

- Each buffer holds the message ID, data, timestamp and flags
- Can vector to one of 20 ISRs on receipt of message
- We will set up one receive and one transmit buffer to share wheel angle and torque information across the bus



FlexCAN Buffer Structure

Activates the buffer and indicates status



0 1 2 3	4 . . . 7	8	9	10	11	12 13 14 15	16 . . . 23	24 . . . 31
	CODE		SRR	IDE	RTR	LENGTH	TIME STAMP	
	ID Standard / Extended [28:18]					ID Extended [17:0]		
Data Byte 0	Data Byte 1						Data Byte 2	Data Byte 3
Data Byte 4	Data Byte 5						Data Byte 6	Data Byte 7

CODE value for receive (Rx) Buffers

Before	After	Description
0000	-	Buffer not active
0010	0010	Buffer is full
0100	0100	Buffer is active and empty
0110	0110	Overrun
0101	0010	An empty buffer was filled
0011	0110	A full/overrun buffer was filled

CODE value for transmit (Tx) Buffers

Before	After	Description
1000	-	Buffer not ready
1100	1000	Buffer ready to Tx
1100	0100	Remote frame will be Tx; Buffer becomes Rx
1010	1010	Data frame will Tx

FlexCAN Buffer Structure

Field	**Description**
Substitute Remote Request (SRR)	Fixed recessive bit, used only in extended format. - Tx buffers: It must be set to ‘1’ by the user. - Rx buffers: It will be stored as received.
ID Extended (IDE)	Should be set to ‘1’ for extended frames, ‘0’ for standard.
Remote Transmission Request (RTR)	0: Current MB has a data frame to be transmitted 1: Current MB has a remote frame to be transmitted
LENGTH	Length of data in bytes.
TIME STAMP	Contains a copy of the free running timer, captured for Tx and Rx frames at the time when the beginning of the Identifier fields appears on the CAN bus.
Frame Identifier (ID)	<u>Standard format</u>: 11 most significant bits (28 to 18) are used, others are ignored. <u>Extended format</u>: all bits are used.
DATA	Up to 8 bytes of data.

Transmission

- Transmit Process: CPU prepares a buffer for transmission by:
  - Write Control/Status word to deactivate TX buffer (code = 1000)
  - Write ID High and Low
  - Write data to be transmitted into TX buffer
  - Write Control/Status word to activate buffer (code 1100), and TX length

Reception

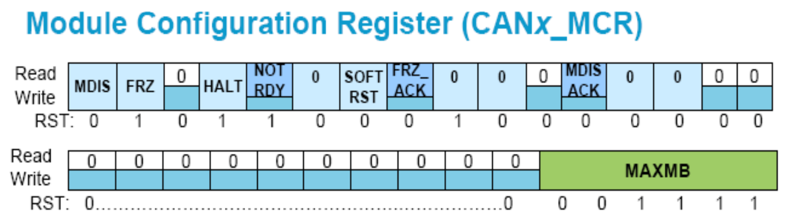
- Receive Process: CPU prepares a buffer for reception by:
  - Write Control/Status word to deactivate RX buffer (code =0000)
  - Write ID High and Low
  - Write Control/Status word to activate RX buffer (code =0100)

Reading the Receive Message Buffer

- Activation of RX buffer causes the following to occur upon a frame reception:
  - Frame is transferred to the first (lowest entry) matching RX message buffer.
  - Value of the free running timer is written into the message buffer time stamp field
  - ID field, Data field (8 bytes at most), and receive length are stored.
  - Code field is updated, (Status flag is set in IFLAG register).
- The CPU reads a receive frame from the message buffer in the order of:
  - Control/Status word (mandatory, as activates the internal lock for this buffer)
  - ID (optional)
  - Data field words
  - Free-running timer (optional)

Additional FlexCAN Special Purpose Registers

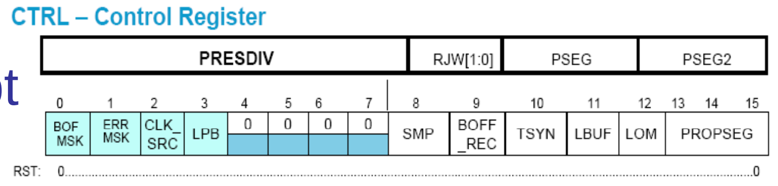
- Module Configuration Register (MCR)
  - Selects mode (normal, debug, low power)
- Control Register (CTL)
  - Establishes CAN clock according to Bosch specification for bus synchronization
- Interrupt Mask Register and Interrupt Flag Register
  - Establishes ISR vector for received messages
- Error and Status Flag Register
- See Chapter 22 in the Reference Manual and `flexcan.h/c`



Module Configuration Register (CANx_MCR)

Read Write	MDIS	FRZ	0	HALT	NOT RDY	0	SOFT RST	FRZ_ ACK	0	0	0	MDIS ACK	0	0	0	0
RST:	0	1	0	1	1	0	0	0	1	0	0	0	0	0	0	0

Read Write	0	0	0	0	0	0	0	0	0	0	MAXMB					
RST:	0									0	0	0	1	1	1	1



CTRL – Control Register

PRESDIV	RJW[1:0]	PSEG	PSEG2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	BOF MSK	ERR MSK	CLK_ SRC	LPB	0	0	0	0	SMP	BOFF _REC	TSYN	LBUF	LOM	PROPSEG		
RST:	0															0

FlexCAN Error & Status Register ESR)

0 1	2	3	4	5	6	7	8	9	10 11	12	13	14	15
BITERR[1:0]	ACKERR	CRCERR	FORMERR	STUFFERR	TXWARN	RXWARN	IDLE	TX/$\overline{RX}$	FCS[1:0]	0	BOFFINT	ERRINT	WAKEINT

RST: 0……….0

THE ERROR CONDITIONS ARE THOSE SINCE THE LAST TIME THIS REGISTER WAS READ. A READ CLEARS ALL ERROR BITS.

BITERR[1:0]: Transmit bit error

- Used to identify type of transmit errors
- 00 = No transmit bit error
- 01 = At least one bit sent as dominant; received as recessive
- 10 = At least one bit sent as recessive; received as dominant
- 11 = Not Used

ACKERR: Acknowledge Error

- 0: No ACK error since last read
- 1: At least one ACK error since last read

CRCERR: Cyclic Redundancy Check Error

- 0: No CRC error in last message
- 1: CRC error in last message

FORMERR: Message Format Error

- 0: No format error in last message
- 1: Format error in last message

STUFFERR: Bit Stuff Error

- 0: No bit errors in last message
- 1: Bit stuffing errors in last message

TXWARN: Transmit Error Status Flag

- 0: Transmit error counter < 96
- 1: Transmit error counter > 96

RXWARN: Receive Error Status Flag

- 0: Receive error counter < 96
- 1: Receive error counter > 96

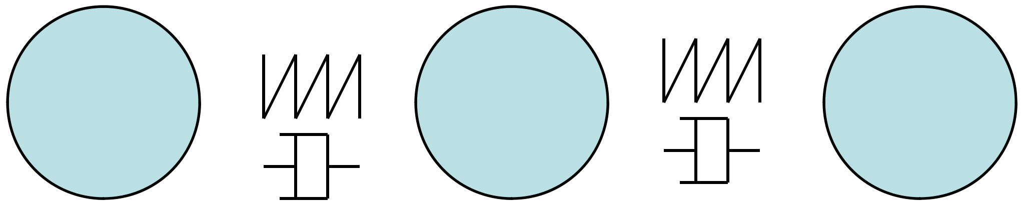
Lab 7: Software

- Good news: FlexCAN driver software has been written for you for Lab 7:
  - See `flexcan.h` and `flexcan.c`
  - Lab documentation describes the sequence of operations required
- You need to write the software for the virtual spring-wall and virtual chain
  - Use, DEC, FQD, PWM from previous labs

Lab 7: Virtual Spring-Wall Software

- DEC ISR of User A reads its wheel angle in degrees as a 32-bit float (use a 1kHz interrupt frequency) and transmits a 4-byte wheel angle message onto the CAN bus.
- User B's CAN message-received ISR is called when User A's message is received
  - Extracts the wheel angle
  - Calculates the virtual spring-wall torque in N-mm
  - Transmits a 4-byte message with the torque value
- User A's CAN message-received ISR updates the motor torque with the value received from User B.

Lab 7: Virtual Chain Software



ω_i, θ_i ω, θ ω_j, θ_j

- Each wheel does this:
  - Read the wheel angle.
  - Estimate the wheel velocity.
  - Transmit an 8-byte message with the wheel angle and velocity, each as 32-bit float values.
  - Compute the torque based on the most recent wheel angles and velocities from the two neighboring lab stations.
  - Update the motor torque.