Blaupunkt (DMS ??)

for controlling a Blaupunkt car radio.

It is basically a 2 wire (rx/tx) async. serial protocol with 9 bits of data where the 8th bit is used for synchronisation.
That made it easy to interface it to a player or PC because you can use the serial port.
The only documents that're left is one sheet of paper containing the initial communication between a cd changer and the radio and the source code.

Here is the protocol cut:

radio	direction,info	changer
	baudrate 4800	
0x17B	(3 times) ->	no response
0x17C	(3 times) ->	no response
0x17D	(3 times) ->	no response
0x17E	(3 times) ->	no response
0x17F	(3 times) ->	no response
0x180	->	0x180
0x48	->	0x48
0x02	->	0x02
0x14F	->, change in baudrate to 9600	no response
0x180	->	0x180
0x9F	->	0x9F
0x14F	->	no response
0x180	->	0x180
0xA1	->	0xA1
0x14F	->	no response
0x180	->	0x180
0xAD	->	0xAD
0x14F	->	no response
0x180	->	0x180
0x48	->	0x48
0x01	->	0x01
0x14F	->	no response
0x10F	<-	0x10F
0x48	<-	0x48
0x01	<-	0x01
0x14F	<-	0x14F
0x103	<-	0x103

0x20	<-	0x20
0x09	<-	0x09
0x20	<-	0x20
0x00	<-	0x00
0x14F	<-	0x14F
0x10B	<-(text info ???)	0x10B
0x20	<- (8 times space)	0x20
0x14F	<-	0x14F
0x101	<-(disc / track info ???)	0x101
0x09	<-	0x09
0x01	<-	0x01
0x14F	<-	0x14F
0x10D	<- (disc / track / time info (BCD) ??)	0x10D
0x01	<-	0x01
0x09	<-	0x09
0x43	<-	0x43
0x57	<-	0x57
0x14F	<-	0x14F

Kenwood

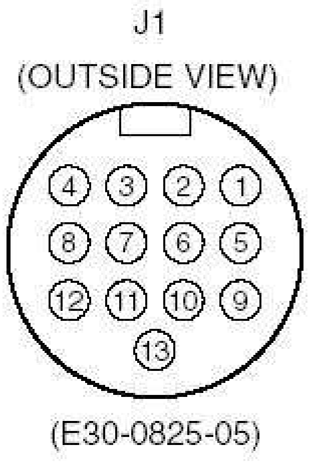
The protocoll used here is a synchron serial protocoll.
First let us start with the connector pinout.

The pins have a 2.54mm distance, so you can simply build a plug using some prototyping board ....

New connector pin-out (for head units >'99?) Thanks to Patrick Loef for this information.

pin	direction	description
1	O	CH-REQH - Request output to changer; "Low" : Request
2	-	Ground
3	-	Vcc +12V
4	O	CH-CON - Changer control; "High" : Operation mode "Low" : Standby
5	I	CH-MUTE - Mute request from changer; "High" : Mute
6	-	AGND - Audio Ground
7	O	CH-RST - Reset output to changer
8	I	Audio right channel

9	I	CH-REQC - Request input from changer; "Low" : Request
10	I	CH-DATAC - Data input from changer
11	O	CH-DATAH - Data output to changer
12	I	Audio left channel
13	I/O	CH_CLK - Clock input/output for changer



1	REQH
2	GND
3	BU+14
4	CH CON
5	CH MUTE
6	A GND
7	CH RST
8	R ch
9	REQC
10	DATAC
11	DATAH
12	L ch
13	CH CLK

The following works only with newer kenwood radios.
Older models have the same pinout but use some more simple protocol ...

The clock low and high periods had a length of 4us.
The data is transfered in bytes (8 bits ... MSB first), data is valid at the rising clock edge..
The data transfer is initiated either by the radio or the changer, the initiator just pulls its fs line low.
When the changer starts the communication it gets 40 clocks from the radio (4 bytes addr + 1byte data size).
The radio then sets its fs to low if it accepts the transfer.
When a transfer is initiated by the radio by setting its fs low it waits for the changer to answer with a low fs,
then it sends the 4 byte addr header, the size byte for the data and the data.

Packet header, direction: both		
byte	log value (r->cdc)	description
0	0x29	destination address
1	0x10	destination address
2	0x1E	own address
3	0x00	own address
4	x	data size

		in bytes
5	x	first data byte
4+data size	x	last data byte

From this point I only write the data part of a packet

initialisation handshake answer, direction: cdc->r		
byte	log value	description
0	0x11	command identifier
1	0xA4	cycle numer of the above packet
2	0x00	??
3	0x01	??
4	0x02	??

send after above packet, maybe radio identification and caps, direction: r->cdc		
byte	log value	description
0	0x20	command identifier
1	0x00	
2	0x11	
3	0x01	
4	0x03	
5	0x0B	
6	0x0B	
7	0x07	
8	0x05	
9	0x83	
10	0x84	
11	0xC0	
12	0xC1	
13	0xC2	
14	0xC3	
15	0xC4	
16	0xC5	

17	0xC6	

send after above packet, maybe init ack from radio, direction: r->cdc		
byte	log value	description
0	0x20	command identifier
1	0x01	
2	0x11	
3	0x29	changer address
4	0x10	changer address
5	0x00	maybe last bytes of cmd 0x11(cdc->r)
6	0x01	maybe last bytes of cmd 0x11(cdc->r)
7	0x02	maybe last bytes of cmd 0x11(cdc->r)

changer caps info, send after above packet, direction: cdc->r		
byte	log value	description
0	0x70	command identifier
1	0x02	
2	0x0A	maybe disc count
3	0x3F	
4	0x03	
5	0x0C	
6	0x02	

play position info, direction: cdc->r		
byte	log value	description
0	0x60	command identifier
1	0x02	maybe sub command id
2	0x00	
3	0x00	
4	0x00	error code, 0 is no error
5	0x00	changer status (load, eject, ...)
6	0x02	play status (1 - play, 2 - pause)

7	0x00	
8	0x01	
9	0x00	track order mode (normal 0, tscan 1, dscan 2,random 6, ...)
10	0x04	
11	0xBB	some bcd number field, displayed when field 3 != 0
12	0x01	
13	0x0B	track number
14	0x07	disc number
15	0x01	min (bcd)
16	0x22	sec (bcd)
17	0x62	min disc (bcd)
18	0x26	sec disc (bcd)
19	0x09	min remain (bcd)
20	0x30	sec remain (bcd)

text info request, direction: r->cdc		
byte	log value	description
0	0x42	command identifier
1	0x02	
2	0x07	disc number
3	0x0A	track number
4	0x00	text section number, sections had 12 bytes size here
5	0x00	
6	0x80	text id (0x80 -> name 0x81 -> artist)

text info send after request, direction: cdc->r		
byte	log value	description
0	0x62	command identifier
1	0x02	
2	0x07	disc number
3	0x02	

4	0x0A	track number (0 -> disc title transfer)
5	0x00	text section number, sections had 12 bytes size here
6	0x09	
7	0x00	
8	0x80	text id (0x80 -> name 0x81 -> artist)
9..20	x	text

commands send when keys on the radio were pressed, direction: r->cdc						
byte	log value (play)	fwd (toggle)	bwd (toggle)	disc- (toggle)	disc+ (toggle)	description
0	0x50	0x50,0x50	0x50,0x50	0x50,0x50	0x50,0x50	command identifier
1	0x02	0x01,0x04	0x01,0x04	0x02,0x00	0x02,0x00	maybe event id (0 all up, 01 dow toggle, 04 up, 06 hold)
2	0x02	0x02,0x02	0x02,0x02	0x02,0x02	0x02,0x02	
3	0x00	0x02,0x02	0x02,0x02	0x00,0x00	0x00,0x00	
4	0x07	0x01,0x01	0x02,0x02	0x04,0x00	0x02,0x00	key id
5	0x00	0x05,0x05	0x06,0x06	0x00,0x00	0x00,0x00	

Using the information above you should have some starting point if you are intrested in doing your own project, it is simple to build a converter to send and receive these commands using a pc so you can find out the meaning of other commands and fields if you need.

Pioneer

The pioneer IP bus uses a 2 wire differential signal for communication.
An equal level on both lines is a logical low while a high is encoded as a voltage difference of some 100mV.
I think a CANbus tranceiver should work here.
The data transfer is initiated by either the cd changer or the radio.The initiator generates a high pulse (ca. 170us) and a following low pulse (
Then the data transfer starts, a 1 is encoded as a high-low sequence with a duration of ap. 20us for both levels and a 0 consists of a 33us hig
The data is now transfered in bytes with MSB first, the 8th bit is an odd parity bit.At the end of the 3rd and all following Bytes there is an addi
receiver acknowledges the transfer.
This is done by holding the data lines in a high state after the initiator sets them low.If this ack is missing the transfer is stopped.

The timings may vary because the real data is encoded in the pulse to space length relation.

The first 3Bytes seem to be some kind of device address.The changer I used transfered a 0x88,0x68,0x00 here while the radio sended 0x88,0
The next 4 bits were always high. After that a size byte and then size bytes were transfered. The last byte in the transfer is a checksum gener
with the 4bit sequence (= 0x0F).

In the following part I only will write the raw data excluding size and cheksum field.

Each command transfered was first answered by some acknowledge packet consisting of a single 0xA1.
(which looks like: 0x88 0x08 0x06 0xF 0x02 0xA1 0xB2 -> 0xB0 is the checksum).

For now I just figured out some very basic things like the fields where time, track and disc number are encoded and also some key codes the radio sends. There are many more fields in the packets where i still don't know the meaning of.
(I just got the radio from a friend for some days and so I couldn't do so much more on it ... however .. if somebody is intrested in some more information and is wiling giving me a radio and a changer for some weeks I'll try to do some more ....)
I have also designed a small circuit using a AT90S2313 controller which can be used for logging the transfer through the pc serial port and als to send commands.

The following packet sended by the changer contained the time disc and track information.

Byte	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Info	command					modus		mcd		disc	min	sec	track		
Data	0x61	0x10	0x06	0x01	0x20	0x04	0x16	0x01	0x06	0x01	0x00	0x00	0x01	0x00	0x

modus:

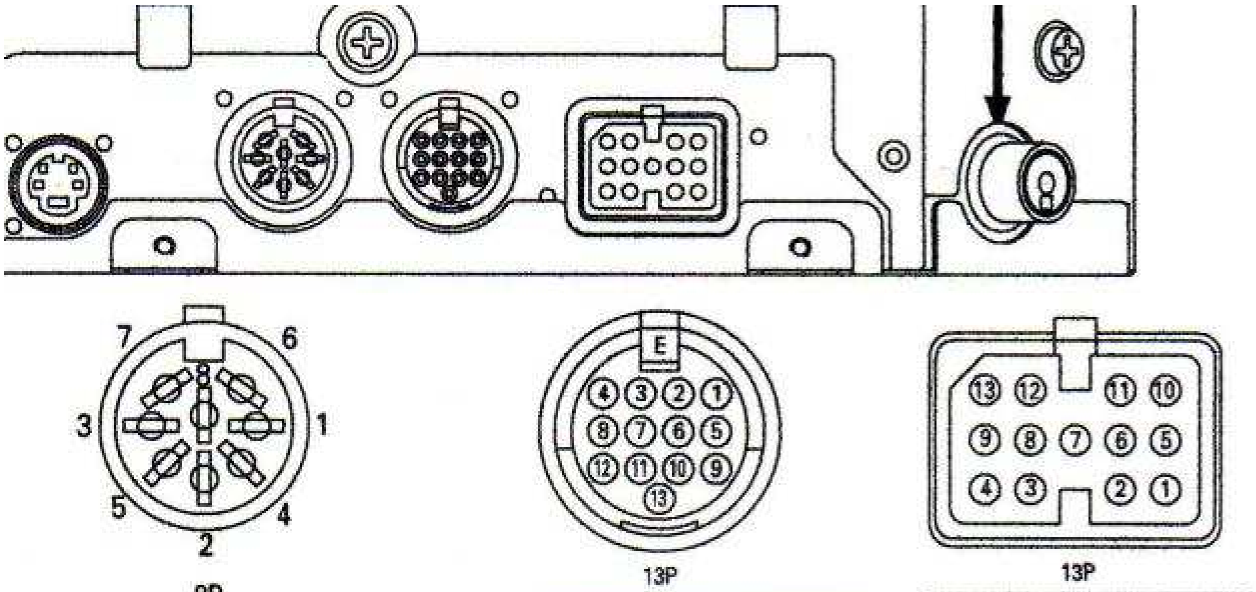
Value	0x02	0x07	0x08	0x10	0x11	0x13	0x14	0x15	0x16
Info	ready	track blink	pause	ready and disc blink	disc blink	load and disc blink	eject and disc blink	load	eject

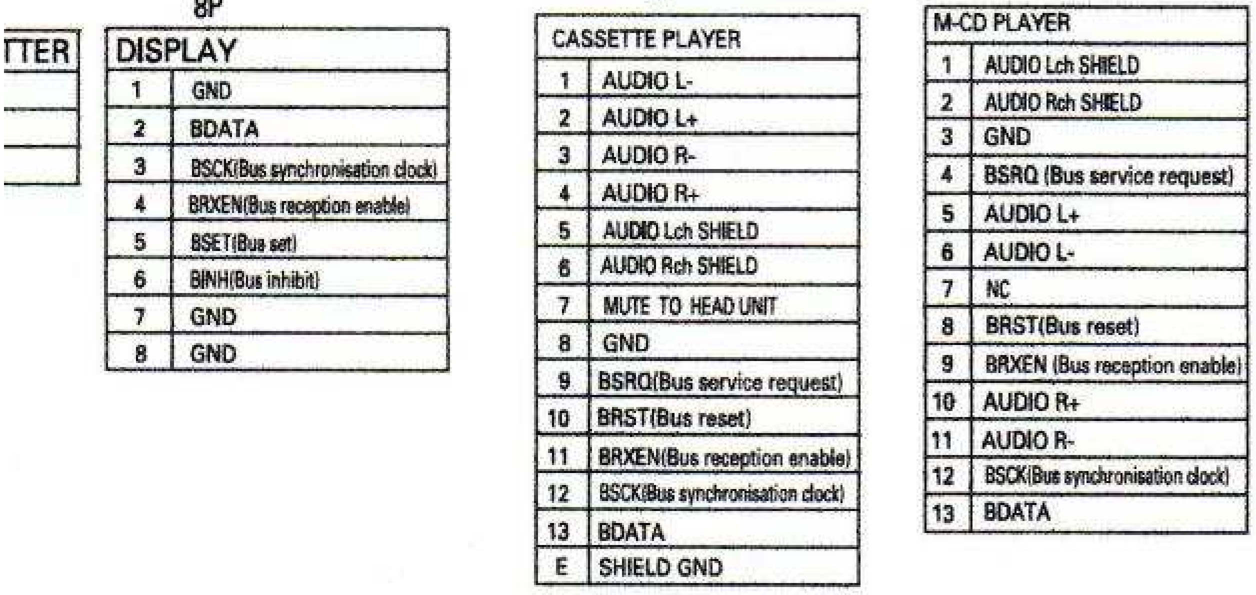
cdt: bit0: (1:cdtext),(0:normal)

The text information was encoded within this packet

Byte	0	1	2	3	4	5	6	7	8	9	10	11	12	13-22
Info	command					modus				disc	track	text seqence number		text
Data	0x61	0x10	0x06	0x01	0x20	0x04	0x38	0x09	0x00	0x06	0x00	0x00	0x00	0x00

Possibly pinout of Pioneer headunit in Renault Espace III





TTER

DISPLAY	
1	GND
2	BDATA
3	BSCK(Bus synchronisation clock)
4	BRXEN(Bus reception enable)
5	BSET(Bus set)
6	BINH(Bus inhibit)
7	GND
8	GND

CASSETTE PLAYER	
1	AUDIO L-
2	AUDIO L+
3	AUDIO R-
4	AUDIO R+
5	AUDIO Lch SHIELD
6	AUDIO Rch SHIELD
7	MUTE TO HEAD UNIT
8	GND
9	BSRQ(Bus service request)
10	BRST(Bus reset)
11	BRXEN(Bus reception enable)
12	BSCK(Bus synchronisation clock)
13	BDATA
E	SHIELD GND

M-CD PLAYER	
1	AUDIO Lch SHIELD
2	AUDIO Rch SHIELD
3	GND
4	BSRQ (Bus service request)
5	AUDIO L+
6	AUDIO L-
7	NC
8	BRST(Bus reset)
9	BRXEN (Bus reception enable)
10	AUDIO R+
11	AUDIO R-
12	BSCK(Bus synchronisation clock)
13	BDATA

Panasonic

The protocol panasonic uses is of the serial sync. type. There is one data line, a clock line and a sync line the changer uses to send data to the
The radio to changer communication is done by some signals known from standard IR remote controls (without a carrier) using one dataline.
This remote control signal is pulse width modulated,the dataline is active high.
After an initail high(9ms) low(4.5ms) there follows a 32 bit sequence with a 0 encoded as 550us high,550us low and a 1 as 550us high,1.7ms
If the low pulse in the init phase is only 2.25ms long it is just a signal send periodical when a key is hold down and there are no data bits.
The data is transfered lsb first, the 1st byte is 0xFF-0th byte and the 3rd byte is 0xFE-2nd byte.The 2nd byte is the command.

The changer to radio communication transfers the data in bytes msb first, the data is valid at the falling clock edge and a low pulse of one half
byte of the transfer on the sync line..The clock period is arround 8us.

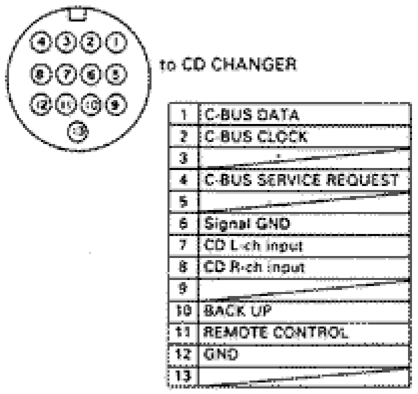
There was only one packet containing state, time, disc and track information.

Byte	0	1	2	3	4	5	6	7

Info		disc(b0-b3)	track	min	sec	state		
Data	0xCB	0x42	0x09	0x02	0x56	0x00	0x30	0xC3

state:

Value	0x00	0x10	0x20	0x04	0x08
Info	normal	scan	random	random	repeat



1	C-BUS DATA
2	C-BUS CLOCK
3	
4	C-BUS SERVICE REQUEST
5	
6	Signal GND
7	CD L-ch input
8	CD R-ch input
9	
10	BACK UP
11	REMOTE CONTROL
12	GND
13	

This should be the pinout for the Clarion 13-pin DIN connector which is used for Clarion C-Bus.

Ford ACP

Ford ACP is a network protocol used by the Head Unit to communicate with and control audio devices such as the Ford 6 disc CD Changer and Telematics units.

It is based on RS485 with 9 bit character data at 9600 baud.

A MAX-481 low power RS485 transceiver will work as interface between a serial USART and ACP bus.

```
+-+  +---------+  +-+
|  1  2  3  4  5  6 |
| [] [] []   [] [] |
|                   |
|  7  8  9 10 11 12 |
| [] [] [] [] [] [] |
+-------------------+
```

Pin Function
1 ACP +
2 ACP Shield
3 GND
4 n/c
5 Audio Left +
6 Audio Right +

7 ACP -
8 CDENABLE
9 +12V Power (unfused)
10 Audio Shield
11 Audio Left -
12 Audio Right +

You will need an AMP plug to connect to the head unit.
AMP Multilock Series 40 cable connector housing with 12 pins or sockets.

The CDENABLE line is 0V when the radio is off and +10V when it is on and can be used as a standby switch for the yampp.
It is not a power supply and can't drive a relay directly.

Communication

* a delay of 1642us (16 Bit times) will indicate a start of new message
* the 9th bit in a byte must be set in the last byte of message to indicate the end of message
* Acknowledge is given with 0x06

Byte 0 - Medium/Priority, should be 0x71

Byte 1 - Changer functional address, should be 0x9A or 0x9B

Byte 2 - Head unit address, 0x80 on receive, 0x82 on transmit

Byte 3 - Command control byte

- 0xE0 - Handshake 1, byte 4 should be 0x04
- 0xFC - Handshake 2, byte 4 must be the same for transmit and receive
- 0xC8 - Handshake 3, byte 4 must be the same for transmit and receive
- 9xFF - Current disc status in byte 4
    - Byte 4 - 0x00 Disk OK
    - Byte 4 - 0x01 No disc in current slot
    - Byte 4 - 0x02 No disc at all
    - Byte 4 - 0x03 Check current disk
    - Byte 4 - 0x04 Check all disc
- 0xC2 and 0xD0 - Change or request current disc
    - Byte 1 - 0x9A - command to change disc
    - Byte 1 - not 0x9A - request current disc
    - Byte 4 - disc number
- 0xC1 - Control command
    - Byte 4
        - Bit 0 - Fast search
        - Bit 1
        - Bit 3
        - Bit 4 - change Random status
        - Bit 5 - change Loudness status
        - Bit 6 - change Play/Stop status
        - Bit 7
    - Send back byte 4 with actual mode
- 0xC3 - Next track
    - Byte 4 - Track number
- 0x43 - Previous track
    - Byte 4 - Track number

The last byte in all message is a checksum of all previous bytes. Simply add all bytes of messag

Message examples
To display current play time, disc and track number:

Byte 0	1	2	3	4	5	6	
0x71	0x9B	0x82	0xD0	Disc No	Track No	Minutes	S

No disc message:

Byte 0	1	2	3	4
0x71	0x9B	0x82	0xFF	0x01

All informations are given without guarantee. Please mail for update or change requests.