

Anexos

Índice

Anexos	1
A - El nRF24L01+	5
A.1 Registros e instrucciones nRF24L01+	5
B - Código de la primera versión	13
B.1 Código del director	13
B.1.1 note_cipher.py	13
B.1.2 nrf24l01.py	15
B.1.3 director.py	19
B.2 Código de los músicos	22
B.2.1 Config.h	22
B.2.2 quartet_main.c	22
C - Código de la segunda versión	35
C.1 Código del director	35
C.1.1 director.py	35
C.2 Código de los músicos	39
C.2.1 Config.h	39
C.2.2 quartet_main.c	39

A - El nRF24L01+

A.1 Registros e instrucciones nRF24L01+

A continuación se muestran las páginas del *datasheet* del nRF24L01+ donde se encuentran las definiciones de los registros y el juego de instrucciones.

9 Register Map

You can configure and control the radio by accessing the register map through the SPI.

9.1 Register map table

All undefined bits in the table below are redundant. They are read out as '0'.

Note: Addresses 18 to 1B are reserved for test purposes, altering them makes the chip malfunction.

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
00	CONFIG				Configuration Register
	Reserved	7	0	R/W	Only '0' allowed
	MASK_RX_DR	6	0	R/W	Mask interrupt caused by RX_DR 1: Interrupt not reflected on the IRQ pin 0: Reflect RX_DR as active low interrupt on the IRQ pin
	MASK_TX_DS	5	0	R/W	Mask interrupt caused by TX_DS 1: Interrupt not reflected on the IRQ pin 0: Reflect TX_DS as active low interrupt on the IRQ pin
	MASK_MAX_RT	4	0	R/W	Mask interrupt caused by MAX_RT 1: Interrupt not reflected on the IRQ pin 0: Reflect MAX_RT as active low interrupt on the IRQ pin
	EN_CRC	3	1	R/W	Enable CRC. Forced high if one of the bits in the EN_AA is high
	CRCO	2	0	R/W	CRC encoding scheme '0' - 1 byte '1' - 2 bytes
	PWR_UP	1	0	R/W	1: POWER UP, 0:POWER DOWN
	PRIM_RX	0	0	R/W	RX/TX control 1: PRX, 0: PTX
01	EN_AA Enhanced ShockBurst™				Enable 'Auto Acknowledgment' Function Disable this functionality to be compatible with nRF2401, see page 72
	Reserved	7:6	00	R/W	Only '00' allowed
	ENAA_P5	5	1	R/W	Enable auto acknowledgement data pipe 5
	ENAA_P4	4	1	R/W	Enable auto acknowledgement data pipe 4
	ENAA_P3	3	1	R/W	Enable auto acknowledgement data pipe 3
	ENAA_P2	2	1	R/W	Enable auto acknowledgement data pipe 2
	ENAA_P1	1	1	R/W	Enable auto acknowledgement data pipe 1
	ENAA_P0	0	1	R/W	Enable auto acknowledgement data pipe 0
02	EN_RXADDR				Enabled RX Addresses
	Reserved	7:6	00	R/W	Only '00' allowed
	ERX_P5	5	0	R/W	Enable data pipe 5.
	ERX_P4	4	0	R/W	Enable data pipe 4.
	ERX_P3	3	0	R/W	Enable data pipe 3.
	ERX_P2	2	0	R/W	Enable data pipe 2.

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
	ERX_P1	1	1	R/W	Enable data pipe 1.
	ERX_P0	0	1	R/W	Enable data pipe 0.
03	SETUP_AW				Setup of Address Widths (common for all data pipes)
	Reserved	7:2	000000	R/W	Only '000000' allowed
	AW	1:0	11	R/W	RX/TX Address field width '00' - Illegal '01' - 3 bytes '10' - 4 bytes '11' - 5 bytes LSByte is used if address width is below 5 bytes
04	SETUP_RETR				Setup of Automatic Retransmission
	ARD ^a	7:4	0000	R/W	Auto Retransmit Delay '0000' - Wait 250µS '0001' - Wait 500µS '0010' - Wait 750µS '1111' - Wait 4000µS (Delay defined from end of transmission to start of next transmission) ^b
	ARC	3:0	0011	R/W	Auto Retransmit Count '0000' - Re-Transmit disabled '0001' - Up to 1 Re-Transmit on fail of AA '1111' - Up to 15 Re-Transmit on fail of AA
05	RF_CH				RF Channel
	Reserved	7	0	R/W	Only '0' allowed
	RF_CH	6:0	0000010	R/W	Sets the frequency channel nRF24L01+ operates on
06	RF_SETUP				RF Setup Register
	CONT_WAVE	7	0	R/W	Enables continuous carrier transmit when high.
	Reserved	6	0	R/W	Only '0' allowed
	RF_DR_LOW	5	0	R/W	Set RF Data Rate to 250kbps. See RF_DR_HIGH for encoding.
	PLL_LOCK	4	0	R/W	Force PLL lock signal. Only used in test
	RF_DR_HIGH	3	1	R/W	Select between the high speed data rates. This bit is don't care if RF_DR_LOW is set. Encoding: [RF_DR_LOW, RF_DR_HIGH]: '00' - 1Mbps '01' - 2Mbps '10' - 250kbps '11' - Reserved

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
	RF_PWR	2:1	11	R/W	Set RF output power in TX mode '00' – -18dBm '01' – -12dBm '10' – -6dBm '11' – 0dBm
	Obsolete	0			Don't care
07	STATUS				Status Register (In parallel to the SPI command word applied on the MOSI pin, the STATUS register is shifted serially out on the MISO pin)
	Reserved	7	0	R/W	Only '0' allowed
	RX_DR	6	0	R/W	Data Ready RX FIFO interrupt. Asserted when new data arrives RX FIFO ^c . Write 1 to clear bit.
	TX_DS	5	0	R/W	Data Sent TX FIFO interrupt. Asserted when packet transmitted on TX. If AUTO_ACK is activated, this bit is set high only when ACK is received. Write 1 to clear bit.
	MAX_RT	4	0	R/W	Maximum number of TX retransmits interrupt Write 1 to clear bit. If MAX_RT is asserted it must be cleared to enable further communication.
	RX_P_NO	3:1	111	R	Data pipe number for the payload available for reading from RX_FIFO 000-101: Data Pipe Number 110: Not Used 111: RX FIFO Empty
	TX_FULL	0	0	R	TX FIFO full flag. 1: TX FIFO full. 0: Available locations in TX FIFO.
08	OBSERVE_TX				Transmit observe register
	PLOS_CNT	7:4	0	R	Count lost packets. The counter is overflow protected to 15, and discontinues at max until reset. The counter is reset by writing to RF_CH . See page 72 .
	ARC_CNT	3:0	0	R	Count retransmitted packets. The counter is reset when transmission of a new packet starts. See page 72 .
09	RPD				
	Reserved	7:1	000000	R	
	RPD	0	0	R	Received Power Detector. This register is called CD (Carrier Detect) in the nRF24L01. The name is different in nRF24L01+ due to the different input power level threshold for this bit. See section 6.4 on page 24 .
0A	RX_ADDR_P0	39:0	0xE7E7E7E7	R/W	Receive address data pipe 0. 5 Bytes maximum length. (LSByte is written first. Write the number of bytes defined by SETUP_AW)

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
0B	RX_ADDR_P1	39:0	0xC2C2C2C2C2C2	R/W	Receive address data pipe 1. 5 Bytes maximum length. (LSByte is written first. Write the number of bytes defined by SETUP_AW)
0C	RX_ADDR_P2	7:0	0xC3	R/W	Receive address data pipe 2. Only LSB. MSBytes are equal to RX_ADDR_P1[39:8]
0D	RX_ADDR_P3	7:0	0xC4	R/W	Receive address data pipe 3. Only LSB. MSBytes are equal to RX_ADDR_P1[39:8]
0E	RX_ADDR_P4	7:0	0xC5	R/W	Receive address data pipe 4. Only LSB. MSBytes are equal to RX_ADDR_P1[39:8]
0F	RX_ADDR_P5	7:0	0xC6	R/W	Receive address data pipe 5. Only LSB. MSBytes are equal to RX_ADDR_P1[39:8]
10	TX_ADDR	39:0	0xE7E7E7E7E7E7	R/W	Transmit address. Used for a PTX device only. (LSByte is written first) Set RX_ADDR_P0 equal to this address to handle automatic acknowledge if this is a PTX device with Enhanced ShockBurst™ enabled. See page 72 .
11	RX_PW_P0				
	Reserved	7:6	00	R/W	Only '00' allowed
	RX_PW_P0	5:0	0	R/W	Number of bytes in RX payload in data pipe 0 (1 to 32 bytes). 0 Pipe not used 1 = 1 byte ... 32 = 32 bytes
12	RX_PW_P1				
	Reserved	7:6	00	R/W	Only '00' allowed
	RX_PW_P1	5:0	0	R/W	Number of bytes in RX payload in data pipe 1 (1 to 32 bytes). 0 Pipe not used 1 = 1 byte ... 32 = 32 bytes
13	RX_PW_P2				
	Reserved	7:6	00	R/W	Only '00' allowed
	RX_PW_P2	5:0	0	R/W	Number of bytes in RX payload in data pipe 2 (1 to 32 bytes). 0 Pipe not used 1 = 1 byte ... 32 = 32 bytes
14	RX_PW_P3				
	Reserved	7:6	00	R/W	Only '00' allowed

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
	RX_PW_P3	5:0	0	R/W	Number of bytes in RX payload in data pipe 3 (1 to 32 bytes). 0 Pipe not used 1 = 1 byte ... 32 = 32 bytes
15	RX_PW_P4				
	Reserved	7:6	00	R/W	Only '00' allowed
	RX_PW_P4	5:0	0	R/W	Number of bytes in RX payload in data pipe 4 (1 to 32 bytes). 0 Pipe not used 1 = 1 byte ... 32 = 32 bytes
16	RX_PW_P5				
	Reserved	7:6	00	R/W	Only '00' allowed
	RX_PW_P5	5:0	0	R/W	Number of bytes in RX payload in data pipe 5 (1 to 32 bytes). 0 Pipe not used 1 = 1 byte ... 32 = 32 bytes
17	FIFO_STATUS				FIFO Status Register
	Reserved	7	0	R/W	Only '0' allowed
	TX_REUSE	6	0	R	Used for a PTX device Pulse the <code>rfce</code> high for at least 10µs to Reuse last transmitted payload. TX payload reuse is active until <code>w_TX_PAYLOAD</code> or <code>FLUSH TX</code> is executed. <code>TX_REUSE</code> is set by the SPI command <code>REUSE_TX_PL</code> , and is reset by the SPI commands <code>w_TX_PAYLOAD</code> or <code>FLUSH TX</code>
	TX_FULL	5	0	R	TX FIFO full flag. 1: TX FIFO full. 0: Available locations in TX FIFO.
	TX_EMPTY	4	1	R	TX FIFO empty flag. 1: TX FIFO empty. 0: Data in TX FIFO.
	Reserved	3:2	00	R/W	Only '00' allowed
	RX_FULL	1	0	R	RX FIFO full flag. 1: RX FIFO full. 0: Available locations in RX FIFO.
	RX_EMPTY	0	1	R	RX FIFO empty flag. 1: RX FIFO empty. 0: Data in RX FIFO.

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
N/A	ACK_PLD	255:0	X	W	Written by separate SPI command ACK packet payload to data pipe number PPP given in SPI command. Used in RX mode only. Maximum three ACK packet payloads can be pending. Payloads with same PPP are handled first in first out.
N/A	TX_PLD	255:0	X	W	Written by separate SPI command TX data payload register 1 - 32 bytes. This register is implemented as a FIFO with three levels. Used in TX mode only.
N/A	RX_PLD	255:0	X	R	Read by separate SPI command. RX data payload register. 1 - 32 bytes. This register is implemented as a FIFO with three levels. All RX channels share the same FIFO.
1C					
	DYNPD				Enable dynamic payload length
	Reserved	7:6	0	R/W	Only '00' allowed
	DPL_P5	5	0	R/W	Enable dynamic payload length data pipe 5. (Requires EN_DPL and ENAA_P5)
	DPL_P4	4	0	R/W	Enable dynamic payload length data pipe 4. (Requires EN_DPL and ENAA_P4)
	DPL_P3	3	0	R/W	Enable dynamic payload length data pipe 3. (Requires EN_DPL and ENAA_P3)
	DPL_P2	2	0	R/W	Enable dynamic payload length data pipe 2. (Requires EN_DPL and ENAA_P2)
	DPL_P1	1	0	R/W	Enable dynamic payload length data pipe 1. (Requires EN_DPL and ENAA_P1)
	DPL_P0	0	0	R/W	Enable dynamic payload length data pipe 0. (Requires EN_DPL and ENAA_P0)
1D					
	FEATURE			R/W	Feature Register
	Reserved	7:3	0	R/W	Only '00000' allowed
	EN_DPL	2	0	R/W	Enables Dynamic Payload Length
	EN_ACK_PAY ^d	1	0	R/W	Enables Payload with ACK
	EN_DYN_ACK	0	0	R/W	Enables the W_TX_PAYLOAD_NOACK command

- Please take care when setting this parameter. If the ACK payload is more than 15 byte in 2Mbps mode the ARD must be 500µS or more, if the ACK payload is more than 5byte in 1Mbps mode the ARD must be 500µS or more. In 250kbps mode (even when the payload is not in ACK) the ARD must be 500µS or more.
- This is the time the PTX is waiting for an ACK packet before a retransmit is made. The PTX is in RX mode for a minimum of 250µS, but it stays in RX mode to the end of the packet if that is longer than 250µS. Then it goes to standby-I mode for the rest of the specified ARD. After the ARD it goes to TX mode and then retransmits the packet.
- The RX_DR IRQ is asserted by a new packet arrival event. The procedure for handling this interrupt should be: 1) read payload through SPI, 2) clear RX_DR IRQ, 3) read FIFO_STATUS to check if there are more payloads available in RX FIFO, 4) if there are more data in RX FIFO, repeat from step 1).

Command name	Command word (binary)	# Data bytes	Operation
R_REGISTER	000A AAAA	1 to 5 LSByte first	Read command and <i>status</i> registers. AAAAA = 5 bit Register Map Address
W_REGISTER	001A AAAA	1 to 5 LSByte first	Write command and <i>status</i> registers. AAAAA = 5 bit Register Map Address Executable in power down or standby modes only.
R_RX_PAYLOAD	0110 0001	1 to 32 LSByte first	Read RX-payload: 1 – 32 bytes. A read operation always starts at byte 0. Payload is deleted from FIFO after it is read. Used in RX mode.
W_TX_PAYLOAD	1010 0000	1 to 32 LSByte first	Write TX-payload: 1 – 32 bytes. A write operation always starts at byte 0 used in TX payload.
FLUSH_TX	1110 0001	0	Flush TX FIFO, used in TX mode
FLUSH_RX	1110 0010	0	Flush RX FIFO, used in RX mode Should not be executed during transmission of acknowledge, that is, acknowledge package will not be completed.
REUSE_TX_PL	1110 0011	0	Used for a PTX device Reuse last transmitted payload. TX payload reuse is active until W_TX_PAYLOAD or FLUSH TX is executed. TX payload reuse must not be activated or deactivated during package transmission.
R_RX_PL_WID ^a	0110 0000	1	Read RX payload width for the top R_RX_PAYLOAD in the RX FIFO.
W_ACK_PAYLOAD ^a	1010 1PPP	1 to 32 LSByte first	Used in RX mode. Write Payload to be transmitted together with ACK packet on PIPE PPP. (PPP valid in the range from 000 to 101). Maximum three ACK packet payloads can be pending. Payloads with same PPP are handled using first in - first out principle. Write payload: 1– 32 bytes. A write operation always starts at byte 0.
W_TX_PAYLOAD_NO_ACK ^a	1011 0000	1 to 32 LSByte first	Used in TX mode. Disables AUTOACK on this specific packet.
NOP	1111 1111	0	No Operation. Might be used to read the <i>STATUS</i> register

a. The bits in the *FEATURE* register shown in [Table 27. on page 60](#) have to be set.

Table 19. Command set for the nRF24L01+ SPI

The *W_REGISTER* and *R_REGISTER* commands operate on single or multi-byte registers. When accessing multi-byte registers read or write to the MSBit of LSByte first. You can terminate the writing before all bytes in a multi-byte register are written, leaving the unwritten MSByte(s) unchanged. For example, the LSByte of *RX_ADDR_P0* can be modified by writing only one byte to the *RX_ADDR_P0* register. The content of the *status* register is always read to *MISO* after a high to low transition on *CSN*.

Note: The 3 bit pipe information in the *STATUS* register is updated during the *IRQ* pin high to low transition. The pipe information is unreliable if the *STATUS* register is read during an *IRQ* pin high to low transition.

B - Código de la primera versión

B.1 Código del director

B.1.1 note_cipher.py

```
1 import pyb
2
3 class NoteCipher:
4
5     def __init__(self, i1, i2, i3, i4):
6
7         self.comptador = 0
8
9         self.identificador = 0x80
10
11         pyb.LED(4).on()
12
13         self.ram1 = self.traduir_partitura(self.llegir_partitura(i1))
14         self.ram2 = self.traduir_partitura(self.llegir_partitura(i2))
15         self.ram3 = self.traduir_partitura(self.llegir_partitura(i3))
16         self.ram4 = self.traduir_partitura(self.llegir_partitura(i4))
17
18         self.trama = self.genera_trama()
19
20         pyb.LED(4).off()
21
22
23     def llegir_partitura(self, i):
24
25         parsed_lines = []
26
27         try:
28             i = open(i, 'r')
29             lines_i = i.readlines()
30             lines_i = filter(lambda x: not x.isspace(), lines_i)
31             parsed_lines=[line.strip()[:-1] if 'Partitura' not in line else
32             line.strip() for line in lines_i]
33             i.close()
```

```

34         return parsed_lines
35
36     except OSError: # catch exception (Input/Output Error)
37         pyb.LED(1).on() # Fire RED Led if file not found
38         print('{0} could not be found'.format(i))
39         pass
40
41     def traduir_partitura(self, partitura):
42         ram = []
43         traduccio_notes = {'c0':0, 'c0_':1, 'd0':2, 'd0_':3, 'e0':4, 'f0':5,
44                             'f0_':6, 'g0':7, 'g0_':8, 'a0':9, 'a0_':10, 'b0':11, 'c1':12,
45                             'c1_':13, 'd1':14, 'd1_':15, 'e1':16, 'f1':17, 'f1_':18, 'g1':19,
46                             'g1_':20, 'a1':21, 'a1_':22, 'b1':23, 'c2':24, 'c2_':25, 'd2':26,
47                             'd2_':27, 'e2':28, 'f2':29, 'f2_':30, 'g2':31, 'g2_':32, 'a2':33,
48                             'a2_':34, 'b2':35, 'c3':36, 'c3_':37, 'd3':38, 'd3_':39, 'e3':40,
49                             'f3':41, 'f3_':42, 'g3':43, 'g3_':44, 'a3':45, 'a3_':46, 'b3':47,
50                             'c4':48, 'c4_':49, 'd4':50, 'd4_':51, 'e4':52, 'f4':53, 'f4_':54,
51                             'g4':55, 'g4_':56}
52         try:
53             for cmd in partitura:
54                 if cmd[:5] == 'play(':
55                     cmd=cmd.replace('
56                                     ',').replace('play(',').replace(')',')')
57                     note, dur = cmd.split(',')
58                     ram.append(int(dur)-1 << 5 | traduccio_notes[note])
59                 elif cmd[:5] == 'pause':
60                     cmd = cmd[5:]
61                     ram.append(0x80 | (int(cmd[1]) - 1))
62                 elif cmd[:8] == 'playmore':
63                     cmd = cmd [8:]
64                     ram.append(0x90 | (int(cmd[1]) - 1))
65                 elif cmd[:7] == 'repeat(':
66                     cmd = cmd[6:]
67                     ram.append(0xC0 | (int(cmd[1])))
68                 elif cmd[:12] == 'repeatmarker':
69                     ram.append(0xC0)
70                 elif cmd[:7] == 'setbase':
71                     cmd = cmd[7:]
72                     ram.append(0xA0 | int(cmd[1]) * 12)
73                 elif cmd[:4] == 'stop':
74                     ram.append(0xE0)
75             except:
76                 pass
77         return ram
78
79     def genera_trama(self):
80
81         max_length = max([len(p) for p in
82                             [self.ram1,self.ram2,self.ram3,self.ram4]])
83
84         equal_lengths = [p+[0]*(max_length-len(p)) for p in
85                             [self.ram1,self.ram2,self.ram3,self.ram4]]

```

```

76
77         return [[equal_lengths[0][i],equal_lengths[1][i],equal_lengths[2][i],
78                 equal_lengths[3][i]] for i in range(max_length)]

```

B.1.2 nrf24l01.py

```

1  """NRF24L01 driver for Micro Python
2  """
3
4  import pyb
5
6  # nRF24L01+ registers
7  CONFIG      = const(0x00)
8  EN_AA       = const(0x01)
9  EN_RXADDR   = const(0x02)
10 SETUP_AW    = const(0x03)
11 SETUP_RETR  = const(0x04)
12 RF_CH       = const(0x05)
13 RF_SETUP    = const(0x06)
14 STATUS      = const(0x07)
15 RX_ADDR_PO  = const(0x0a)
16 TX_ADDR     = const(0x10)
17 RX_PW_PO    = const(0x11)
18 FIFO_STATUS = const(0x17)
19 DYNPD       = const(0x1c)
20
21 # CONFIG register
22 EN_CRC       = const(0x08) # enable CRC
23 CRCO         = const(0x04) # CRC encoding scheme; 0=1 byte, 1=2 bytes
24 PWR_UP       = const(0x02) # 1=power up, 0=power down
25 PRIM_RX      = const(0x01) # RX/TX control; 0=PTX, 1=PRX
26
27 # RF_SETUP register
28 POWER_0      = const(0x00) # -18 dBm
29 POWER_1      = const(0x02) # -12 dBm
30 POWER_2      = const(0x04) # -6 dBm
31 POWER_3      = const(0x06) # 0 dBm
32 SPEED_1M     = const(0x00)
33 SPEED_2M     = const(0x08)
34 SPEED_250K   = const(0x20)
35
36 # STATUS register
37 RX_DR        = const(0x40) # RX data ready; write 1 to clear
38 TX_DS        = const(0x20) # TX data sent; write 1 to clear
39 MAX_RT       = const(0x10) # max retransmits reached; write 1 to clear
40
41 # FIFO_STATUS register
42 RX_EMPTY     = const(0x01) # 1 if RX FIFO is empty
43
44 # constants for instructions
45 R_RX_PL_WID = const(0x60) # read RX payload width
46 R_RX_PAYLOAD = const(0x61) # read RX payload

```

```

47 W_TX_PAYLOAD = const(0xa0) # write TX payload
48 FLUSH_TX     = const(0xe1) # flush TX FIFO
49 FLUSH_RX     = const(0xe2) # flush RX FIFO
50 NOP          = const(0xff) # use to read STATUS register
51
52 class NRF24L01:
53     def __init__(self, spi, cs, ce, channel=64, payload_size=1):
54         assert payload_size <= 32
55
56         # init the SPI bus and pins
57         spi.init(spi.MASTER, baudrate=65625, polarity=0, phase=0, firstbit=spi.MSB)
58         cs.init(cs.OUT_PP, cs.PULL_NONE)
59         ce.init(ce.OUT_PP, ce.PULL_NONE)
60
61         # store the pins
62         self.spi = spi
63         self.cs = cs
64         self.ce = ce
65
66         # reset everything
67         self.ce.low()
68         self.cs.high()
69         self.payload_size = payload_size
70         self.pipe0_read_addr = None
71         pyb.delay(5)
72
73         # set address width to 5 bytes and check for device present
74         self.reg_write(SETUP_AW, 0b11)
75         if self.reg_read(SETUP_AW) != 0b11:
76             raise OSError("nRF24L01+ Hardware not responding")
77
78         # disable dynamic payloads
79         self.reg_write(DYNPD, 0)
80
81         # auto retransmit delay: 1750us
82         # auto retransmit count: 8
83         self.reg_write(SETUP_RETR, (6 << 4) | 8)
84
85         # set rf power and speed
86         self.set_power_speed(POWER_3, SPEED_1M) # Best for point to point links
87
88         # init CRC
89         self.set_crc(2)
90
91         # clear status flags
92         self.reg_write(STATUS, RX_DR | TX_DS | MAX_RT)
93
94         # set channel
95         self.set_channel(channel)
96
97         # flush buffers
98         self.flush_rx()
99         self.flush_tx()

```



```
100
101 def reg_read(self, reg):
102     self.cs.low()
103     self.spi.send_recv(reg)
104     buf = self.spi.recv(1)
105     self.cs.high()
106     return buf[0]
107
108 def reg_write(self, reg, buf):
109     self.cs.low()
110     status = self.spi.send_recv(0x20 | reg)[0]
111     self.spi.send(buf)
112     self.cs.high()
113     return status
114
115 def flush_rx(self):
116     self.cs.low()
117     self.spi.send(FLUSH_RX)
118     self.cs.high()
119
120 def flush_tx(self):
121     self.cs.low()
122     self.spi.send(FLUSH_TX)
123     self.cs.high()
124
125 # power is one of POWER_x defines; speed is one of SPEED_x defines
126 def set_power_speed(self, power, speed):
127     setup = self.reg_read(RF_SETUP) & 0b11010001
128     self.reg_write(RF_SETUP, setup | power | speed)
129
130 # length in bytes: 0, 1 or 2
131 def set_crc(self, length):
132     config = self.reg_read(CONFIG) & ~(CRCO | EN_CRC)
133     if length == 0:
134         pass
135     elif length == 1:
136         config |= EN_CRC
137     else:
138         config |= EN_CRC | CRCO
139     self.reg_write(CONFIG, config)
140
141 def set_channel(self, channel):
142     self.reg_write(RF_CH, min(channel, 125))
143
144 # address should be a bytes object 5 bytes long
145 def open_tx_pipe(self, address):
146     assert len(address) == 5
147     self.reg_write(RX_ADDR_P0, address)
148     self.reg_write(TX_ADDR, address)
149     self.reg_write(RX_PW_P0, self.payload_size)
150
151 # address should be a bytes object 5 bytes long
152 # pipe 0 and 1 have 5 byte address
```

```

153 # pipes 2-5 use same 4 most-significant bytes as pipe 1, plus 1 extra byte
154
155 def open_rx_pipe(self, pipe_id, address):
156     assert len(address) == 5
157     assert 0 <= pipe_id <= 5
158     if pipe_id == 0:
159         self.pipe0_read_addr = address
160     if pipe_id < 2:
161         self.reg_write(RX_ADDR_P0 + pipe_id, address)
162     else:
163         self.reg_write(RX_ADDR_P0 + pipe_id, address[0])
164     self.reg_write(RX_PW_P0 + pipe_id, self.payload_size)
165     self.reg_write(EN_RXADDR, self.reg_read(EN_RXADDR) | (1 << pipe_id))
166     # for correct use of AA on pipe 0
167     if self.reg_read(EN_AA) & (1 << pipe_id) == (1 << pipe_id):
168         self.reg_write(TX_ADDR, address)
169
170 def start_listening(self):
171     self.reg_write(CONFIG, self.reg_read(CONFIG) | PWR_UP | PRIM_RX)
172     self.reg_write(STATUS, RX_DR | TX_DS | MAX_RT)
173
174     if self.pipe0_read_addr is not None:
175         self.reg_write(RX_ADDR_P0, self.pipe0_read_addr)
176
177     self.flush_rx()
178     self.flush_tx()
179     self.ce.high()
180     pyb.udelay(130)
181
182 def stop_listening(self):
183     self.ce.low()
184     self.flush_tx()
185     self.flush_rx()
186
187 # returns True if any data available to recv
188 def any(self):
189     return not bool(self.reg_read(FIFO_STATUS) & RX_EMPTY)
190
191 def recv(self):
192     # get the data
193     self.cs.low()
194     self.spi.send(R_RX_PAYLOAD)
195     buf = self.spi.recv(self.payload_size)
196     self.cs.high()
197     # clear RX ready flag
198     self.reg_write(STATUS, RX_DR)
199
200     return buf
201
202 # blocking wait for tx complete
203 def send(self, buf, timeout=500):
204     send_nonblock = self.send_start(buf)
205     start = pyb.millis()

```

```

206     result = None
207     while result is None and pyb.elapsed_millis(start) < timeout:
208         result = self.send_done() # 1 == success, 2 == fail
209     if result == 2:
210         raise OSError("send failed")
211
212     # non-blocking tx
213     def send_start(self, buf):
214         # power up
215         self.reg_write(CONFIG, (self.reg_read(CONFIG) | PWR_UP) & ~PRIM_RX)
216         pyb.udelay(150)
217         # send the data
218         self.cs.low()
219         self.spi.send(W_TX_PAYLOAD)
220         self.spi.send(buf)
221         if len(buf) < self.payload_size:
222             self.spi.send(b'\x00' * (self.payload_size - len(buf))) # pad out data
223         self.cs.high()
224
225         # enable the chip so it can send the data
226         self.ce.high()
227         pyb.udelay(15) # needs to be >10us
228         self.ce.low()
229
230     # returns None if send still in progress, 1 for success, 2 for fail
231     def send_done(self):
232         if not (self.reg_read(STATUS) & (TX_DS | MAX_RT)):
233             return None # tx not finished
234
235         # either finished or failed: get and clear status flags, power down
236         status = self.reg_write(STATUS, RX_DR | TX_DS | MAX_RT)
237         self.reg_write(CONFIG, self.reg_read(CONFIG) & ~PWR_UP)
238         return 1 if status & TX_DS else 2

```

B.1.3 director.py

```

1  import struct
2  import pyb
3  from pyb import Pin, SPI
4  from nrf24l01_driver import NRF24L01
5  from note_cipher import NoteCipher
6
7  START_ID      = const(0x01)
8  NOTE_ID       = const(0x23)
9  TEMPO_ID      = const(0x48)
10 STATUS_ID     = const(0x76)
11 ALIVE         = const(0x98)
12 # Bass TX_ADDR
13 #b'\x11\x11\x11\x11\x11'
14
15 # Viololin TX_ADDR
16 #b'\x41\x41\x41\x41\x41'

```

```

17
18 # Guitar_1 TX_ADDR
19 #b'\x71\x71\x71\x71\x71'
20
21 # Guitar_2 TX_ADDR
22 #b'\xd1\xd1\xd1\xd1\xd1'
23
24 class Conductor:
25
26     def __init__(self, tempo=32, direccion_rx=b'\xb2\xb2\xb3\xb4\x01',
27                 direcciones_tx=[b'\x11\x11\x11\x11\x11', b'\x41\x41\x41\x41\x41',
28                                 b'\x71\x71\x71\x71\x71', b'\xd1\xd1\xd1\xd1\xd1']):
29
30         self.tempo = tempo
31         self.direccion_rx = direccion_rx
32
33         self.num_inst = len(direcciones_tx)
34         self.direcciones_tx = direcciones_tx
35
36         self.nrf = NRF24L01(SPI(1), Pin('X5'), Pin('X4'), channel=64,
37                             payload_size=2)
38
39         self.note_cipher = NoteCipher('partituras/i1.txt', 'partituras/i2.txt',
40                                       'partituras/i3.txt', 'partituras/i4.txt')
41
42     def check_all_alive(self):
43
44         instruments_alive = [0]*self.num_inst
45         ask_status = struct.pack('BB', STATUS_ID, 0)
46
47         while instruments_alive != [1]*self.num_inst:
48             for instrument_tx_dir in self.direcciones_tx:
49                 if instruments_alive[self.direcciones_tx.index(instrument_tx_dir)] == 0:
50                     try:
51                         self.nrf.open_tx_pipe(instrument_tx_dir)
52                         self.nrf.send(ask_status)
53                         print("asking instrument on address: ", instrument_tx_dir)
54                     except OSError:
55                         print("Error sending message to instrument on address",
56                               instrument_tx_dir)
57                     pass
58                 self.nrf.open_rx_pipe(0, self.direccion_rx)
59                 pyb.udelay(140)
60                 self.nrf.start_listening()
61                 while True:
62                     if self.nrf.any():
63                         while self.nrf.any():
64                             buf = self.nrf.recv()
65                             message_id, data = struct.unpack('BB', buf)
66                             if message_id == STATUS_ID and data==ALIVE:
67                                 instruments_alive[self.direcciones_tx.index(instrument_tx_dir)]
68                                     = 1

```

```
66         print(instruments_alive)
67         print("instrument alive on address", instrument_tx_dir)
68         pyb.udelay(15)
69         self.nrf.stop_listening()
70         break
71         pyb.delay(10)
72     print("All instruments alive")
73     return True
74
75     def send_data_to_all(self,message_id,content):
76
77         set_tempo = struct.pack('BB',message_id,content)
78
79         for instrument_tx_dir in self.direcciones_tx:
80             try:
81                 self.nrf.open_tx_pipe(instrument_tx_dir)
82                 self.nrf.send(set_tempo)
83                 print(message_id,content," sent to instrument on address: ",
84                       instrument_tx_dir)
85             except OSError:
86                 print("Error sending message to instrument on address",
87                       instrument_tx_dir)
88                 pass
89             return
90
91     def send_notelist(self,instrument_tx):
92
93         notelist_index = self.direcciones_tx.index(instrument_tx)
94         notelist = []
95         for trama in self.note_cipher.trama:
96             notelist+=[trama[notelist_index]]
97         for note in notelist:
98             note_pack = struct.pack('BB',NOTE_ID,note)
99             try:
100                 self.nrf.open_tx_pipe(instrument_tx)
101                 self.nrf.send(note_pack)
102                 pyb.delay(10)
103             except OSError:
104                 print("Failed to send note to address",instrument_tx)
105                 pass
106         pyb.delay(10)
107         return
108
109     def main(self):
110
111         while not self.check_all_alive():
112             pass
113         pyb.udelay(40)
114         self.send_data_to_all(TEMPO_ID,self.tempo)
115         pyb.udelay(40)
116         for instrument_tx in self.direcciones_tx:
117             self.send_notelist(instrument_tx)
118             pyb.udelay(40)
```

```

117     self.send_data_to_all(START_ID,0x00)
118
119     if __name__ == '__main__':
120         p=Conductor()
121         p.main()

```

B.2 Código de los músicos

B.2.1 Config.h

```

1  // #define Bass
2  // #define Violin
3  // #define Guitar1
4  // #define Guitar2
5
6  #if defined(Bass)&& defined(Violin)
7      #error "Only one instrument allowed"
8  #endif
9
10 #if defined(Bass)&& defined(Guitar1)
11     #error "Only one instrument allowed"
12 #endif
13
14 #if defined(Guitar1)&& defined(Violin)
15     #error "Only one instrument allowed"
16 #endif
17
18 #define START_ID 0x01
19 #define NOTA_ID 0x23
20 #define TEMPO_ID 0x48
21 #define STATUS_ID 0x76

```

B.2.2 quartet_main.c

```

1  #include <p18f4520.h>
2  #include "OSA.h"
3  #include "OSAcfg.h"
4  #include "sinus.h"
5  #include "bach1067.h"
6  #include "UPC_nRF24L01.h"
7  #include "ConfigBits.h"
8
9  #include "Config.h"
10
11
12
13 #define pin_ENABLE_INS 1//PORTAbits.RAO // Switch bass channel ON/OFF
14
15

```

```

16 //-----
17
18 #define DAC_TYPE_PWM 0
19 #define DAC_TYPE_R2R 1
20
21 #define DAC_TYPE DAC_TYPE_PWM
22
23
24 //-----
25
26
27 #define PORTA_CONST 0x00
28 #define TRISA_CONST 0x0F
29 #define PORTC_CONST 0x00
30 #define TRISC_CONST 0b11111011
31
32
33 //*****
34 // Define type for sound variables
35 //*****
36 typedef struct {
37     // Common notelist variables
38     unsigned char cBaseNote; // Lowest note for instrument (lowest octave)
39     unsigned char cCurNote; // Current note position in notelist
40     unsigned char *NoteList; // Pointer to notelist table in ROM (see music.c)
41     char cDuration; // Current note (or pause) duration
42
43     // Control variables
44     unsigned char cRepeatPosition; // Point of start of repeat
45     char bEnable:1; // Sound present or paused
46     char bRepeat:1; // Now repeating fragment
47     char bStopped:1; // Channel stopped (see CMD_STOP in notelists in
48         misc.c)
49
50     // Wave generating variables (see interrupt.c)
51     unsigned int F; // Current signal frequency
52     unsigned int f; // Position of current signal to take value from
53         sinus table (see sinus.c)
54     unsigned char t; // Signal form position (see interrupt: lables
55         FORMING_SIGNAL_XXX)
56 } TSound;
57
58 //*****
59 // Define sound variables
60 //*****
61 TSound S; // For channel
62
63 OST_FLAG flag_Playing; // Each bit in this variable means that channel is playing
64     now.

```

```

64 // When bit becomes "0", it means that end of notelist was
65 // reached.
66 #define FLAG_INS_PLAYING 0x01
67
68
69 ////////////////////////////////////////////////////////////////////
70 // Fast multiply (signed char) * (char) MACRO
71 ////////////////////////////////////////////////////////////////////
72 //define MUL() temp_dac += temp1 * temp2
73
74 #define MUL(); _asm  clrf  temp3, 1 \
75                    btfsc temp1, 7,1 \
76                    comf  temp3, 1,1 \
77                    clrf  WREG,0   \
78                    rrcf  temp3, 1,1 \
79                    rrcf  temp1, 1,1 \
80                    btfsc temp2, 7,1 \
81                    addwf temp1, 0,1 \
82                    rrcf  temp3, 1,1 \
83                    rrcf  temp1, 1,1 \
84                    btfsc temp2, 6,1 \
85                    addwf temp1, 0,1 \
86                    rrcf  temp3, 1,1 \
87                    rrcf  temp1, 1,1 \
88                    btfsc temp2, 5,1 \
89                    addwf temp1, 0,1 \
90                    rrcf  temp3, 1,1 \
91                    rrcf  temp1, 1,1 \
92                    btfsc temp2, 4,1 \
93                    addwf temp1, 0,1 \
94                    rrcf  temp3, 1,1 \
95                    rrcf  temp1, 1,1 \
96                    btfsc temp2, 3,1 \
97                    addwf temp1, 0,1 \
98                    rrcf  temp3, 1,1 \
99                    rrcf  temp1, 1,1 \
100                   btfsc temp2, 2,1 \
101                   addwf temp1, 0,1 \
102                   rrcf  temp3, 1,1 \
103                   rrcf  temp1, 1,1 \
104                   btfsc temp2, 1,1 \
105                   addwf temp1, 0,1 \
106                   movwf temp1,1  _endasm \
107                   temp_dac += temp1;
108
109
110 //*****
111
112 //-----
113 // Interrupt variables
114 //-----
115 char prs;

```



```

116 char m_cDAC;
117
118 signed char temp1;
119 unsigned char temp2, temp3;
120 int temp_dac;
121
122 char min, max;
123
124 unsigned char tempo;
125
126 unsigned char in;
127 unsigned char mensaje_llegada[2];
128 unsigned char mensaje_rebotado[2];
129 unsigned char informe;
130
131 unsigned char direccion_tx[5]={0xB2, 0xB2, 0xB3, 0xB4, 0x01};
132 #ifdef Bass
133     unsigned char direccion_rx[5]={0x11, 0x11, 0x11, 0x11, 0x11};
134 #endif
135 #ifdef Violin
136     unsigned char direccion_rx[5]={0x41, 0x41, 0x41, 0x41, 0x41};
137 #endif
138 #ifdef Guitar1
139     unsigned char direccion_rx[5]={0x71, 0x71, 0x71, 0x71, 0x71};
140 #endif
141 #ifdef Guitar2
142     unsigned char direccion_rx[5]={0xD1, 0xD1, 0xD1, 0xD1, 0xD1};
143 #endif
144
145
146 #pragma udata section = 0x100 // Indicamos al compilador que queremos que la
147 // cola circular este situada a partir
148 // de la direccion 0x100
149 unsigned char cola_rx[256]; // cola circular de 256 posiciones
150 #pragma udata
151
152 //-----
153
154
155 // Interrupt routine (sound syntezier)
156 //#include "interrupt.c"
157 void high_isr(void);
158 //*****
159
160
161
162
163 // Init PIC periphery
164 //*****
165 void Init (void)
166 {
167
168     PORTA = PORTA_CONST;

```

```

169 PORTC = PORTC_CONST;
170 CMCON = 7;
171
172 TRISA = TRISA_CONST;
173 TRISC = TRISC_CONST;
174 T2CON = 0x3C; // prs = 1; post = 8
175 PR2 = 51-1; // TMR2 period = (PR2+1) * prs * post * Tcyc = 64 * 1 * 8 * 0.2 =
           102 us
176 // PWM freq = 1 / ((PR2+1) * prs * Tcyc) = 78 KHz
177
178 #if DAC_TYPE == DAC_TYPE_PWM
179 CCP1CON = 0x0C;
180 #endif
181
182 INTCONbits.PEIE = 1;
183 PIE1bits.TMR2IE = 1;
184 }
185
186
187 //*****
188 // Init instrument variable
189 //*****
190 void InitSoundVariable(TSound *S, unsigned char *notelist)
191 {
192     OS_EnterCriticalSection();
193     S->bEnable = 1; // Make sound ON
194     S->bStopped = 0;
195     S->bRepeat = 0; // Clear repeate fragment flag
196     S->cCurNote = 0; // Play from start of notelist
197     S->NoteList = notelist; // Pointer to notes in ROM (see music.c)
198     S->cDuration = 0; // Clear current duration
199     S->F = 0; // Current signal frequency
200     S->t = 0; // Signal form
201     S->f = 0;
202     OS_LeaveCriticalSection();
203 }
204
205
206
207
208 //*****
209 // Read next note from list and update sound variable
210 //*****
211 // On return: duration of new note (or pause)
212 void NoteWork (TSound *S)
213 {
214     char n, cmd;
215     int f;
216     unsigned char aux2;
217
218     if (S->bStopped) return ; // Do not read notelist if channel stopped
219
220     // We will read notelist until we get next note

```

```

221 do {
222     // Read next value from notelist
223     cmd = S->NoteList[S->cCurNote++];
224
225     // Check: is it note or control command
226     //-----
227     if (cmd & 0x80) { // this is command
228     //-----
229         n = cmd & 0xE0;
230         switch (n) {
231
232             case CMD_PAUSE: // Make pause for several ticks
233                 if (!(cmd & 0x10)) S->bEnable = 0;
234                 n = cmd & 0xF;
235                 cmd = 0;
236                 break;
237             case CMD_SET_BASE: // Set lowes octave
238                 S->cBaseNote = (char)(cmd & 0x1F);
239                 break;
240
241             case CMD_REPEAT: // Repeat fragment
242                 if (S->bRepeat || !(cmd & 0x1F)) { // Set repeat marker
243                     S->bRepeat = 0;
244                     S->cRepeatPosition = S->cCurNote; // (save current position as
// marker)
245                 } else { // Repeat from marker
246                     S->bRepeat = 1;
247                     S->cCurNote = S->cRepeatPosition; // Restore position of marker
248                 }
249                 break;
250
251             case CMD_STOP: // Stop channel
252                 S->bStopped = 1;
253                 n = 0;
254                 cmd = 0;
255                 break;
256         }
257     //-----
258     } else { // this is note
259     //-----
260         aux2 = S->cBaseNote + (cmd & 0x1F);
261         f = Freq[S->cBaseNote + (cmd & 0x1F)]; // Set note frequency
262         OS_EnterCriticalSection(); // Disable interrupts
263         S->bEnable = 1; // Sound ON
264         S->F = f; // Position in sinus table
265         S->t = 0; // Position in signal form
266         OS_LeaveCriticalSection(); // Enable interrupts
267
268         n = cmd >> 4; // bits 5 and 6 - duration
269         n >>= 1;
270         break;
271     }
272     //-----

```

```

273     } while (cmd & 0x80); // all commands have bit7 = 1
274
275     S->cDuration = n;
276 }
277
278 //-----
279
280 void Task_INS (void)
281 {
282     S.bStopped = 1;
283     for (;;) {
284         if (S.bStopped) {
285             // Tell to conductor, that notelist over
286             OS_Flag_Set_0(flag_Playing, FLAG_INS_PLAYING);
287
288             // Wait for command to start playing from conductor
289
290             OS_Bsem_Wait(BS_START_MUSIC);
291
292             // Re-init channel data
293             InitSoundVariable(&S, cola_rx);
294             OS_Flag_Set_1(flag_Playing, FLAG_INS_PLAYING);
295
296             // Retranslate command to next task
297             OS_Bsem_Set(BS_START_MUSIC);
298         }
299
300         // Wait for command from conductor
301         do {
302             OS_Bsem_Wait(BS_INS);
303         } while (S.cDuration--);
304
305         // Read next note and set new duration
306         NoteWork(&S);
307     }
308 }
309
310
311
312 //-----
313
314 void Task_CONDUCTOR (void)
315 {
316     for (;;) {
317         OS_Bsem_Reset(BS_START_MUSIC);
318         if (OS_Flag_Check_00(flag_Playing, 0xFF)) {
319             OS_Bsem_Set(BS_START_MUSIC);
320         };
321
322         OS_Delay(tempo);
323
324         // When all channels got CMD_STOP in notelist, conductor restarts music
325         OS_Bsem_Set(BS_INS);

```

```

326     }
327 }
328 //-----
329
330 void main (void)
331 {
332
333 unsigned char i;
334     Init();           // Init periphery
335     InitSoundVariable(&S, cola_rx);
336     SPI_Start(0b10);
337     nRF24L01_Ports_Start();
338     i=0;
339     while(1)
340     {
341         Start_RX_Mode_nRF24L01(0b11, 0b1000000, 0, 0b11, 1, 1, direccion_rx, 0x00,
342             0x00, 0x00, 0x00, 0x00, 0, 2, 0, 0, 0, 0, 0);
343         informe=Receive_Data_RX_Mode_nRF24L01(0, 100, 2, mensaje_llegada); //No
344             Checksum
345         Finish_nRF24L01_Operation();
346         if(!(informe & 0b00010000))
347         {
348             if(mensaje_llegada[0]==STATUS_ID)
349             {
350                 mensaje_rebotado[0]=STATUS_ID;
351                 mensaje_rebotado[1]=ALIVE;
352                 Start_TX_Mode_nRF24L01(0b11, 0b1000000, 0, 0b11, 1, 1, 0b0011, 15, 0,
353                     2);
354                 Delay100TCYx(100);
355                 Send_Data_TX_Mode_nRF24L01(0,0b11,direccion_tx,2,mensaje_rebotado);
356                 Finish_nRF24L01_Operation();
357                 break;
358             }
359         }
360     }
361     Start_RX_Mode_nRF24L01(0b11, 0b1000000, 0, 0b11, 1, 1, direccion_rx, 0x00, 0x00,
362         0x00, 0x00, 0x00, 0, 2, 0, 0, 0, 0, 0);
363
364     while (1) {
365
366         informe=Receive_Data_RX_Mode_nRF24L01(0, 100, 2, mensaje_llegada); //No
367             Checksum
368         if(!(informe & 0b00010000))
369         {
370             if(mensaje_llegada[0]==TEMPO_ID)
371             {
372                 tempo = mensaje_llegada[1];
373             }
374
375             if(mensaje_llegada[0]==START_ID)
376             {
377                 Nop();

```

```

374         Nop();
375         Nop();
376         Nop();
377         break;
378     }
379
380     if(mensaje_llegada[0] == NOTA_ID)
381     {
382         #ifdef Bass
383             cola_rx[i++] = mensaje_llegada[1];
384         #endif
385
386         #ifdef Violin
387             cola_rx[i++] = mensaje_llegada[1];
388         #endif
389
390         #ifdef Guitar1
391             cola_rx[i++] = mensaje_llegada[1];
392         #endif
393
394         #ifdef Guitar2
395             cola_rx[i++] = mensaje_llegada[1];
396         #endif
397     }
398 }
399 }
400 }
401
402 OS_Init();    // Init system variables
403
404 S.bEnable = 0;
405
406 // Create all tasks
407 OS_Task_Create(0, Task_INS);
408
409 OS_Task_Create(1, Task_CONDUCTOR); // this task must have lower priority
410
411 min = 0xFF;
412 max = 0x00;
413
414 OS_EI();    // Enable interrupts
415
416
417
418 OS_Run();    // Run OS kernel
419 }
420
421 //-----
422 // ISR
423 //-----
424 /*****High priority interrupt vector *****/
425
426 #pragma code high_vector=0x08

```

```

427 void interrupt_at_high_vector(void)
428 {
429     _asm GOTO high_isr _endasm
430 }
431
432 #pragma code
433 #pragma interrupt high_isr
434 void high_isr (void)
435 {
436     PIR1bits.TMR2IF = 0;
437
438     #if DAC_TYPE == DAC_TYPE_R2R
439         PORTB = m_cDAC;
440     #endif
441     // LATAbits.LATA4 = 1;
442     temp_dac = 0;
443
444     //-----
445     // FOUR SAMPLES SINTEZING
446
447     //-----
448     /***** BASS *****/
449
450
451
452
453 #ifndef Bass
454     if (S.bEnable && pin_ENABLE_INS) {
455
456         // READING SINUS
457         temp1 = bass[*((char*)&S.f+1) & 0x3F];
458
459         // FORMING SIGNAL_BASS
460         if (S.t > 64) {
461             temp2 = S.t >> 1;
462             temp2 = 128 - temp2;
463         } else if (S.t >= 4) {
464             temp2 = 192 - S.t;
465         } else {
466             if (!S.t) {
467                 temp2 = prs << 2;
468             } else {
469                 temp2 = 255;
470                 if (S.t & 2) temp2 &= ~0x20;
471                 if (S.t & 1) temp2 &= ~0x10;
472             }
473         }
474         MUL();
475         S.f += S.F;
476     }
477 #endif
478
479 #ifndef Violin

```

```

480     if (S.bEnable && pin_ENABLE_INS) {
481
482         // READING SINUS
483         temp1 = violin[*((char*)&S.f+1) & 0x3F];
484
485         // FORMING SIGNAL_VIOLIN
486         if (S.t < 4) {
487             temp2 = S.t << 4;
488             temp2 <=& 2;
489         } else {
490             if (!(S.t & 0x80)) {
491                 temp2 = 0xC0;
492             } else {
493                 temp2 = S.t - 0x80;
494                 temp2 = 0xC0 - temp2;
495             }
496         } /**/
497         MUL();
498         S.f += S.F;
499     }
500 #endif
501
502
503 #if defined(Guitar1) || defined(Guitar2)
504     if (S.bEnable && pin_ENABLE_INS) {
505
506         // READING SINUS
507         temp1 = guitar[*((char*)&S.f+1) & 0x3F];
508         // FORMING SIGNAL_GUITAR
509         if (S.t > 64) {
510             temp2 = S.t >> 2;
511             temp2 = 64 - temp2;
512         } else if (S.t >= 4) {
513             temp2 = 128 - S.t;
514         } else {
515             if (!S.t) {
516                 temp2 = prs << 2;
517             } else {
518                 temp2 = 255;
519                 if (S.t & 2) temp2 &= ~0x40;
520                 if (S.t & 1) temp2 &= ~0x20;
521             }
522         }
523         MUL();
524         S.f += S.F;
525     }
526 #endif
527
528
529
530     //-----
531
532     temp_dac >>= 2;

```



```
533     m_cDAC = *((char*)&temp_dac+0) + 0x80; // Next value to out
534
535     if (min > m_cDAC) min = m_cDAC;
536     if (max < m_cDAC) max = m_cDAC;
537
538     #if DAC_TYPE == DAC_TYPE_PWM
539         m_cDAC >>= 2;
540         CCP1CONbits.CCP1X = 0;
541         CCP1CONbits.CCP1Y = 0;
542         if (temp_dac & 2) CCP1CONbits.CCP1X = 1;
543         if (temp_dac & 1) CCP1CONbits.CCP1Y = 1 ;
544         CCPR1L = m_cDAC;
545
546     #endif
547
548     //-----
549     prs++;
550     if (prs & 0x40) {
551         prs = 0;
552
553         if (S.t!=0xFF) S.t ++;
554
555
556         OS_Timer();
557     }
558     //-----
559
560
561     // LATAbits.LATA4 = 0;
562 }
```


C - Código de la segunda versión

C.1 Código del director

C.1.1 director.py

```
1 import struct
2 import pyb
3 from pyb import Pin, SPI
4 from nrf24l01_driver import NRF24L01
5 from note_cipher import NoteCipher
6
7 START_ID      = const(0x01)
8 NOTE_ID       = const(0x23)
9 TEMPO_ID      = const(0x48)
10 STATUS_ID     = const(0x76)
11 ALIVE         = const(0x98)
12 EMPTY        = const(0xAA)
13 FULL         = const(0x55)
14 NO_DATA      = const(0x00)
15 NO_ANSWER    = const(0x17)
16 # Bass TX_ADDR
17 #b'\x11\x11\x11\x11\x11'
18
19 # Vilolin TX_ADDR
20 #b'\x41\x41\x41\x41\x41'
21
22 # Guitar_1 TX_ADDR
23 #b'\x71\x71\x71\x71\x71'
24
25 # Guitar_2 TX_ADDR
26 #b'\xd1\xd1\xd1\xd1\xd1'
27
28 class Conductor:
29
30     def __init__(self, tempo=32,
31                 direccion_rx=b'\xb2\xb2\xb3\xb4\x01', direcciones_tx=[b'\x11\x11\x11\x11\x11',
32                 b'\x41\x41\x41\x41\x41', b'\x71\x71\x71\x71\x71', b'\xd1\xd1\xd1\xd1\xd1']):
```

```

33     self.tempo          = tempo
34     self.direccion_rx  = direccion_rx
35
36     self.num_inst      = len(direcciones_tx)
37     self.direcciones_tx = direcciones_tx
38
39     self.nrf           = NRF24L01(SPI(1), Pin('X5'), Pin('X4'), channel=64,
40                                 payload_size=2)
41
42     self.note_cipher   = NoteCipher('partituras/i1.txt', 'partituras/i2.txt',
43                                   'partituras/i3.txt', 'partituras/i4.txt')
44
45     self.status_array = []
46     self.pointers_array = [0]*self.num_inst
47
48
49     def check_all_alive(self):
50
51         instruments_alive = [0]*self.num_inst
52         ask_status = struct.pack('BB', STATUS_ID, 0)
53
54         while instruments_alive != [ALIVE]*self.num_inst:
55             for instrument_tx_dir in self.direcciones_tx:
56                 if instruments_alive[self.direcciones_tx.index(instrument_tx_dir)] == 0:
57                     try:
58                         self.nrf.open_tx_pipe(instrument_tx_dir)
59                         self.nrf.send(ask_status)
60                         print("asking instrument on address: ", instrument_tx_dir)
61                     except OSError:
62                         print("Error sending message to instrument on address",
63                               instrument_tx_dir)
64                     pass
65                 self.nrf.open_rx_pipe(0, self.direccion_rx)
66                 pyb.udelay(140)
67                 self.nrf.start_listening()
68                 while True:
69                     if self.nrf.any():
70                         while self.nrf.any():
71                             buf = self.nrf.recv()
72                             message_id, data = struct.unpack('BB', buf)
73                             if message_id == STATUS_ID and data == ALIVE:
74                                 instruments_alive[self.direcciones_tx.index(instrument_tx_dir)]
75                                     = ALIVE
76                                 print(instruments_alive)
77                                 print("instrument alive on address", instrument_tx_dir)
78                                 pyb.udelay(15)
79                                 self.nrf.stop_listening()
80                                 break
81                                 pyb.delay(10)
82         print("All instruments alive")
83         self.status_array = instruments_alive
84         return True

```

```
83
84     def send_data_to_all(self,message_id,payload):
85
86         message = struct.pack('BB',message_id,payload)
87
88         for instrument_tx_dir in self.direcciones_tx:
89             try:
90                 self.nrf.open_tx_pipe(instrument_tx_dir)
91                 self.nrf.send(message)
92                 print(message_id,payload," sent to instrument on address: ",
93                       instrument_tx_dir)
94             except OSError:
95                 print("Error sending message to instrument on address",
96                       instrument_tx_dir)
97                 pass
98         return
99
100     def
101     send_data_to_one(self,message_id,payload,instrument_tx,waiting_time=50,wait_for_response
102     = True):
103
104     message = struct.pack('BB',message_id,payload)
105     self.nrf.open_tx_pipe(instrument_tx)
106     try:
107         self.nrf.send(message)
108         print(message_id,payload," sent to instrument on address: ", instrument_tx)
109     except OSError:
110         print("Error sending message to instrument on address", instrument_tx)
111         pass
112
113     if wait_for_response:
114         self.nrf.open_rx_pipe(0, self.direccion_rx)
115         pyb.udelay(140)
116         self.nrf.start_listening()
117         start_time = pyb.millis()
118         timeout = False
119         while not timeout and not self.nrf.any():
120             if pyb.elapsed_millis(start_time) > waiting_time:
121                 timeout = True
122             self.nrf.stop_listening()
123             if not timeout:
124                 buf = self.nrf.recv()
125                 print(self.nrf.any())
126                 print(buf)
127                 m_id,data = struct.unpack('BB', buf)
128                 return data
129             else:
130                 pass
131         return NO_ANSWER
132
133     def send_next_note(self,instrument_tx):
134
135         index = self.direcciones_tx.index(instrument_tx)
```

```

132     note_pointer = self.pointers_array[index]
133     note = self.note_cipher.trama[note_pointer][index]
134     if note != 0:
135         status = self.send_data_to_one(NOTE_ID,note,instrument_tx)
136         if status != NO_ANSWER:
137             self.status_array[index] = status
138             self.pointers_array[index] += 1
139         return
140
141
142     def ask_status(self,instrument_tx):
143         status = self.send_data_to_one(STATUS_ID,NO_DATA,instrument_tx)
144         index = self.direcciones_tx.index(instrument_tx)
145         if status != NO_ANSWER:
146             self.status_array[index] = status
147         return
148
149
150     def send_notelist_n(self,instrument_tx,n):
151
152         notelist_index = self.direcciones_tx.index(instrument_tx)
153         notelist = []
154         for i in range(n):
155             notelist+=self.note_cipher.trama[i][notelist_index]
156         for note in notelist:
157             note_pack = struct.pack('BB',NOTE_ID,note)
158             try:
159                 self.nrf.open_tx_pipe(instrument_tx)
160                 self.nrf.send(note_pack)
161                 pyb.delay(10)
162             except OSError:
163                 print("Failed to send note to address",instrument_tx)
164                 pass
165         self.status_array[self.direcciones_tx.index(instrument_tx)] = EMPTY
166         self.pointers_array[self.direcciones_tx.index(instrument_tx)] = n
167         pyb.delay(10)
168         return
169
170
171     def main(self):
172
173         while not self.check_all_alive():
174             pass
175
176         self.send_data_to_all(TEMPO_ID,self.tempo)
177
178         for instrument_tx in self.direcciones_tx:
179             self.send_notelist_n(instrument_tx,50)
180
181         self.send_data_to_all(START_ID,0x00)
182
183         while True:
184             for direccion_tx in self.direcciones_tx:

```

```

185         index=self.direcciones_tx.index(direccion_tx)
186         if self.status_array[index] == EMPTY:
187             self.send_next_note(direccion_tx)
188         elif self.status_array[index] == FULL:
189             self.ask_status(direccion_tx)
190
191 if __name__ == '__main__':
192     p=Conductor()
193     p.main()

```

C.2 Código de los músicos

C.2.1 Config.h

```

1  //define Bass
2  //define Violin
3  //define Guitar1
4  //define Guitar2
5
6  #if defined(Bass)&& defined(Violin)
7      #error "Only one instrument allowed"
8  #endif
9
10 #if defined(Bass)&& defined(Guitar1)
11     #error "Only one instrument allowed"
12 #endif
13
14 #if defined(Guitar1)&& defined(Violin)
15     #error "Only one instrument allowed"
16 #endif
17
18 #define START_ID 0x01
19 #define NOTA_ID 0x23
20 #define TEMPO_ID 0x48
21 #define STATUS_ID 0x76
22
23 #define ALIVE 0x98
24
25 #define FULL 0x55
26 #define EMPTY 0xAA
27
28 #define QUEUE_MIN 20

```

C.2.2 quartet_main.c

```

1  #include <p18f4520.h>
2  #include "OSA.h"
3  #include "OSAcfg.h"
4  #include "sinus.h"

```

```

5 #include "bach1067.h"
6 #include "UPC_nRF24L01.h"
7 #include <delays.h>
8
9 #include "ConfigBits.h"
10
11 #include "Config.h"
12 #include "iden.h"
13
14
15 #define pin_ENABLE_INS 1//PORTAbits.RA0 // Switch bass channel ON/OFF
16 #define FULL 0x55
17 #define EMPTY 0xAA
18
19 //-----
20
21 #define DAC_TYPE_PWM 0
22 #define DAC_TYPE_R2R 1
23
24 #define DAC_TYPE DAC_TYPE_PWM
25
26
27 //-----
28
29
30 #define PORTA_CONST 0x00
31 #define TRISA_CONST 0x0F
32 #define PORTC_CONST 0x00
33 #define TRISC_CONST 0b11111011
34
35
36 //*****
37 // Define type for sound variables
38 //*****
39 typedef struct {
40     // Common notelist variables
41     unsigned char cBaseNote; // Lowest note for instrument (lowest octave)
42     unsigned char cCurNote; // Current note position in notelist
43     unsigned char *NoteList; // Pointer to notelist table in ROM (see music.c)
44     char cDuration; // Current note (or pause) duration
45
46     // Control variables
47     unsigned char cRepeatPosition; // Point of start of repeat
48     char bEnable:1; // Sound present or paused
49     char bRepeat:1; // Now repeating fragment
50     char bStopped:1; // Channel stopped (see CMD_STOP in notelists in
51         misc.c)
52
53     // Wave generating variables (see interrupt.c)
54     unsigned int F; // Current signal frequency
55     unsigned int f; // Position of current signal to take value from
56         sinus table (see sinus.c)

```



```

55     unsigned char t;           // Signal form position (see interrupt: lables
        FORMING_SIGNAL_XXX)
56
57 } TSound;
58
59
60
61 //*****
62 // Define sound variables
63 //*****
64 TSound S; // For channel
65
66
67 OST_FLAG  flag_Playing; // Each bit in this variable means that channel is playing
        now.
68
        // When bit becomes "0", it means that end of notelist was
        reached.
69
70 #define FLAG_INS_PLAYING    0x01
71
72
73 ////////////////////////////////////////////////////////////////////
74 // Fast multiply (signed char) * (char) MACRO
75 ////////////////////////////////////////////////////////////////////
76 //#define MUL() temp_dac += temp1 * temp2
77
78 #define MUL(); _asm    clrfsf    temp3, 1 \
79                      btfsf    temp1, 7,1 \
80                      comf     temp3, 1,1 \
81                      clrfsf    WREG,0 \
82                      rrcfsf    temp3, 1,1 \
83                      rrcfsf    temp1, 1,1 \
84                      btfsf    temp2, 7,1 \
85                      addwfsf  temp1, 0,1 \
86                      rrcfsf    temp3, 1,1 \
87                      rrcfsf    temp1, 1,1 \
88                      btfsf    temp2, 6,1 \
89                      addwfsf  temp1, 0,1 \
90                      rrcfsf    temp3, 1,1 \
91                      rrcfsf    temp1, 1,1 \
92                      btfsf    temp2, 5,1 \
93                      addwfsf  temp1, 0,1 \
94                      rrcfsf    temp3, 1,1 \
95                      rrcfsf    temp1, 1,1 \
96                      btfsf    temp2, 4,1 \
97                      addwfsf  temp1, 0,1 \
98                      rrcfsf    temp3, 1,1 \
99                      rrcfsf    temp1, 1,1 \
100                     btfsf    temp2, 3,1 \
101                     addwfsf  temp1, 0,1 \
102                     rrcfsf    temp3, 1,1 \
103                     rrcfsf    temp1, 1,1 \
104                     btfsf    temp2, 2,1 \

```

```

105         addwf  temp1, 0,1 \
106         rrcf   temp3, 1,1 \
107         rrcf   temp1, 1,1 \
108         btfsc  temp2, 1,1 \
109         addwf  temp1, 0,1 \
110         movwf  temp1,1   _endasm \
111         temp_dac += temp1;
112
113
114 //*****
115
116 //-----
117 // Interrupt variables
118 //-----
119 char prs;
120 char m_cDAC;
121
122 signed char temp1;
123 unsigned char temp2, temp3;
124 int temp_dac;
125
126 char min, max;
127
128 unsigned char tempo;
129
130
131
132 unsigned char i;
133
134 unsigned char counter = 0; // comptador missatges rebuts
135 unsigned char QUEUE_MAX = 50;
136
137 unsigned char mensaje_llegada[2];
138 unsigned char mensaje_rebotado[2];
139 unsigned char informe;
140
141 unsigned char direccion_tx[5]={0xB2, 0xB2, 0xB3, 0xB4, 0x01};
142 #ifdef Bass
143     unsigned char direccion_rx[5]={0x11, 0x11, 0x11, 0x11, 0x11};
144 #endif
145 #ifdef Violin
146     unsigned char direccion_rx[5]={0x41, 0x41, 0x41, 0x41, 0x41};
147 #endif
148 #ifdef Guitar1
149     unsigned char direccion_rx[5]={0x71, 0x71, 0x71, 0x71, 0x71};
150 #endif
151 #ifdef Guitar2
152     unsigned char direccion_rx[5]={0xD1, 0xD1, 0xD1, 0xD1, 0xD1};
153 #endif
154
155
156 #pragma udata section = 0x100 // Indicamos al compilador que queremos que la
157 // cola circular este situada a partir

```

```

158 // de la direccion 0x100
159 unsigned char cola_rx[256]; // cola circular de 256 posiciones
160 #pragma udata
161
162
163
164
165
166
167 //-----
168
169
170 // Interrupt routine (sound syntezier)
171 // #include "interrupt.c"
172 void high_isr(void);
173 //*****
174
175
176 // Init PIC periphery
177 //*****
178 void Init (void)
179 {
180     PORTA = PORTA_CONST;
181     PORTC = PORTC_CONST;
182     CMCON = 7;
183
184     TRISA = TRISA_CONST;
185     TRISC = TRISC_CONST;
186     T2CON = 0x3C; // prs = 1; post = 8
187     PR2 = 51-1; // TMR2 period = (PR2+1) * prs * post * Tcyc = 64 * 1 * 8 * 0.2 =
188                 // 102 us
189                 // PWM freq = 1 / ((PR2+1) * prs * Tcyc) = 78 KHz
190 #if DAC_TYPE == DAC_TYPE_PWM
191     CCP1CON = 0x0C;
192 #endif
193
194     INTCONbits.PEIE = 1;
195     PIE1bits.TMR2IE = 1;
196 }
197
198
199 //*****
200 // Init instrument variable
201 //*****
202 void InitSoundVariable(TSound *S, unsigned char *notelist)
203 {
204     OS_EnterCriticalSection();
205     S->bEnable = 1; // Make sound ON
206     S->bStopped = 0;
207     S->bRepeat = 0; // Clear repeate fragment flag
208     S->cCurNote = 0; // Play from start of notelist
209     S->NoteList = notelist; // Pointer to notes in ROM (see music.c)

```

```

210     S->cDuration = 0;           // Clear current duration
211     S->F = 0;                 // Current signal frequency
212     S->t = 0;                 // Signal form
213     S->f = 0;
214     OS_LeaveCriticalSection();
215 }
216
217
218 //*****
219 // Read next note from list and update sound variable
220 //*****
221 // On return: duration of new note (or pause)
222 void NoteWork (TSound *S)
223 {
224     char n, cmd;
225     int f;
226     unsigned char aux2;
227
228     if (S->bStopped) return ;    // Do not read notelist if channel stopped
229
230     // We will read notelist until we get next note
231
232     do {
233         // Read next value from notelist
234         cmd = S->NoteList[S->cCurNote++];
235         counter = counter - 1;
236         // Check: is it note or control command
237         //-----
238         if (cmd & 0x80) {        // this is command
239             //-----
240             n = cmd & 0xE0;
241             switch (n) {
242
243             case CMD_PAUSE:     // Make pause for several ticks
244                 if (!(cmd & 0x10)) S->bEnable = 0;
245                 n = cmd & 0xF;
246                 cmd = 0;
247                 break;
248             case CMD_SET_BASE:  // Set lowes octave
249                 S->cBaseNote = (char)(cmd & 0x1F);
250                 break;
251
252             case CMD_REPEAT:    // Repeat fragment
253                 if (S->bRepeat || !(cmd & 0x1F)) { // Set repeat marker
254                     S->bRepeat = 0;
255                     S->cRepeatPosition = S->cCurNote; // (save current position as
256                     // marker)
257                 } else {        // Repeat from marker
258                     S->bRepeat = 1;
259                     S->cCurNote = S->cRepeatPosition; // Restore position of marker
260                 }
261                 break;

```

```

262     case CMD_STOP:    // Stop channel
263         Nop();
264         Nop();
265         Nop();
266         Nop();
267         Nop();
268         Nop();
269         S->bStopped = 1;
270         n = 0;
271         cmd = 0;
272         Reset();
273         break;
274     }
275     //-----
276 } else {           // this is note
277 //-----
278     aux2 = S->cBaseNote + (cmd & 0x1F);
279     f = Freq[S->cBaseNote + (cmd & 0x1F)]; // Set note frequency
280     OS_EnterCriticalSection();           // Disable interrupts
281     S->bEnable = 1;                       // Sound ON
282     S->F = f;                             // Position in sinus table
283     S->t = 0;                             // Position in signal form
284     OS_LeaveCriticalSection();           // Enable interrupts
285
286     n = cmd >> 4;                        // bits 5 and 6 - duration
287     n >>= 1;
288     break;
289 }
290 //-----
291 } while (cmd & 0x80); // all commands have bit7 = 1
292
293 S->cDuration = n;
294 }
295
296 //-----
297
298 void Task_INS (void)
299 {
300     S.bStopped = 1;
301     for (;;) {
302         if (S.bStopped) {
303             // Tell to conductor, that notelist over
304             OS_Flag_Set_0(flag_Playing, FLAG_INS_PLAYING);
305
306             // Wait for command to start playing from conductor
307
308             OS_Bsem_Wait(BS_START_MUSIC);
309
310             // Re-init channel data
311             InitSoundVariable(&S, cola_rx);
312             OS_Flag_Set_1(flag_Playing, FLAG_INS_PLAYING);
313
314             // Retranslate command to next task

```

```

315     OS_Bsem_Set(BS_START_MUSIC);
316     }
317
318     // Wait for command from conductor
319     do {
320         OS_Bsem_Wait(BS_INS);
321     } while (S.cDuration--);
322
323     // Read next note and set new duration
324     NoteWork(&S);
325     }
326 }
327
328
329
330 //-----
331
332 void Task_CONDUCTOR (void)
333 {
334     for (;;) {
335         OS_Bsem_Reset(BS_START_MUSIC);
336         if (OS_Flag_Check_00(flag_Playing, 0xFF)) {
337             OS_Bsem_Set(BS_START_MUSIC);
338         };
339
340         OS_Delay(tempo);
341
342         // When all channels got CMD_STOP in notelist, conductor restarts music
343         OS_Bsem_Set(BS_INS);
344     }
345 }
346 //-----
347
348 void Task_comRF (void)
349 {
350
351     for (;;) {
352         //OS_Yield();
353         Start_RX_Mode_nRF24L01(0b11, 0b1000000, 0, 0b11, 1, 1, direccion_rx, 0x00, 0x00,
354             0x00, 0x00, 0x00, 0, 2, 0, 0, 0, 0, 0);
355
356         //OS_Cond_Wait(!(Receive_Data_RX_Mode_nRF24L01(0, 100, 2, mensaje_llegada)&
357             0b00010000)); // Wait for RF message
358
359         CE=1;
360         OS_Cond_Wait(!IRQ);
361         informe=Receive_Data_RX_Mode_nRF24L01(0, 100, 2, mensaje_llegada); //No Checksum
362         Finish_nRF24L01_Operation();
363         if(!(informe & 0b00010000))
364         {
365             if(mensaje_llegada[0]==NOTA_ID)
366             {
367                 counter = counter + 1;

```

```

366         cola_rx[i++] = mensaje_llegada[1];
367
368         mensaje_rebotado[0]=EMPTY;
369         mensaje_rebotado[1]=EMPTY;
370         if (counter >= QUEUE_MAX)
371         {
372
373             mensaje_rebotado[0]=FULL;
374             mensaje_rebotado[1]=FULL;
375         }
376
377         //mensaje_rebotado[0]=STATUS_ID;
378         Start_TX_Mode_nRF24L01(0b11, 0b1000000, 0, 0b11, 1, 1, 0b0011, 15, 0,
379             2);
380         Delay100TCYx(100);
381         Send_Data_TX_Mode_nRF24L01(0,0b11,direccion_tx,2,mensaje_rebotado);
382         Finish_nRF24L01_Operation();
383     }
384     else if (mensaje_llegada[0] == STATUS_ID)
385     {
386         mensaje_rebotado[1]=FULL;
387         mensaje_rebotado[0]=FULL;
388         if (counter<=QUEUE_MIN)
389         {
390
391             mensaje_rebotado[0]=EMPTY;
392             mensaje_rebotado[1]=EMPTY;
393         }
394
395         Start_TX_Mode_nRF24L01(0b11, 0b1000000, 0, 0b11, 1, 1, 0b0011, 15, 0, 2);
396         Delay100TCYx(100);
397         Send_Data_TX_Mode_nRF24L01(0,0b11,direccion_tx,2,mensaje_rebotado);
398         Finish_nRF24L01_Operation();
399     }
400 }
401 }
402 }
403 }
404 }
405 }
406
407 //-----
408 void main (void)
409 {
410
411     Init();           // Init periphery
412     InitSoundVariable(&S, cola_rx);
413     SPI_Start(0b10);
414     nRF24L01_Ports_Start();
415     i=0;
416     while(1)
417     {

```

```

418 Start_RX_Mode_nRF24L01(0b11, 0b1000000, 0, 0b11, 1, 1, direccion_rx, 0x00,
      0x00, 0x00, 0x00, 0, 2, 0, 0, 0, 0, 0);
419 informe=Receive_Data_RX_Mode_nRF24L01(0, 100, 2, mensaje_llegada); //No
      Checksum
420 Finish_nRF24L01_Operation();
421 if(!(informe & 0b00010000))
422 {
423     if(mensaje_llegada[0]==STATUS_ID)
424     {
425         mensaje_rebotado[0]=STATUS_ID;
426         mensaje_rebotado[1]=ALIVE;
427         Start_TX_Mode_nRF24L01(0b11, 0b1000000, 0, 0b11, 1, 1, 0b0011, 15, 0,
            2);
428         Delay100TCYx(100);
429         Send_Data_TX_Mode_nRF24L01(0,0b11,direccion_tx,2,mensaje_rebotado);
430         Finish_nRF24L01_Operation();
431         break;
432     }
433 }
434 }
435 Start_RX_Mode_nRF24L01(0b11, 0b1000000, 0, 0b11, 1, 1, direccion_rx, 0x00, 0x00,
      0x00, 0x00, 0, 2, 0, 0, 0, 0, 0);
436
437 while (1) {
438
439     informe=Receive_Data_RX_Mode_nRF24L01(0, 100, 2, mensaje_llegada); //No
      Checksum
440
441     if(!(informe & 0b00010000))
442     {
443         if(mensaje_llegada[0]==TEMPO_ID)
444         {
445             tempo = mensaje_llegada[1];
446         }
447
448         if(mensaje_llegada[0]==START_ID)
449         {
450             break;
451         }
452
453         if(mensaje_llegada[0] == NOTA_ID)
454         {
455             counter++;
456             #ifdef Bass
457                 cola_rx[i++] = mensaje_llegada[1];
458             #endif
459
460             #ifdef Violin
461                 cola_rx[i++] = mensaje_llegada[1];
462             #endif
463
464             #ifdef Guitar1
465                 cola_rx[i++] = mensaje_llegada[1];

```



```

466         #endif
467
468         #ifdef Guitar2
469             cola_rx[i++] = mensaje_llegada[1];
470         #endif
471     }
472
473 }
474 }
475
476 OS_Init();    // Init system variables
477
478 S.bEnable = 0;
479
480 // Create all tasks
481 OS_Task_Create(0, Task_INS);
482
483 OS_Task_Create(1, Task_CONDUCTOR);
484
485 OS_Task_Create(2, Task_comRF);
486
487 min = 0xFF;
488 max = 0x00;
489
490
491 OS_EI();    // Enable interrupts
492
493
494
495 OS_Run();    // Run OS kernel
496 }
497
498 //-----
499 // ISR
500 //-----
501 /*****High priority interrupt vector *****/
502
503 #pragma code high_vector=0x08
504 void interrupt_at_high_vector(void)
505 {
506     _asm GOTO high_isr _endasm
507 }
508
509 #pragma code
510 #pragma interrupt high_isr
511 void high_isr (void)
512 {
513     PIR1bits.TMR2IF = 0;
514
515 #if DAC_TYPE == DAC_TYPE_R2R
516     PORTB = m_cDAC;
517 #endif
518 // LATAbits.LATA4 = 1;

```

```

519     temp_dac = 0;
520
521     //-----
522     // FOUR SAMPLES SINTEZING
523
524     //-----
525     /***** BASS *****/
526
527
528
529
530 #ifndef Bass
531     if (S.bEnable && pin_ENABLE_INS) {
532
533         // READING SINUS
534         temp1 = bass[*((char*)&S.f+1) & 0x3F];
535
536         // FORMING SIGNAL_BASS
537         if (S.t > 64) {
538             temp2 = S.t >> 1;
539             temp2 = 128 - temp2;
540         } else if (S.t >= 4) {
541             temp2 = 192 - S.t;
542         } else {
543             if (!S.t) {
544                 temp2 = prs << 2;
545             } else {
546                 temp2 = 255;
547                 if (S.t & 2) temp2 &= ~0x20;
548                 if (S.t & 1) temp2 &= ~0x10;
549             }
550         }
551         MUL();
552         S.f += S.F;
553     }
554 #endif
555
556 #ifndef Violin
557     if (S.bEnable && pin_ENABLE_INS) {
558
559         // READING SINUS
560         temp1 = violin[*((char*)&S.f+1) & 0x3F];
561
562         // FORMING SIGNAL_VIOLIN
563         if (S.t < 4) {
564             temp2 = S.t << 4;
565             temp2 <<= 2;
566         } else {
567             if (!(S.t & 0x80)) {
568                 temp2 = 0xC0;
569             } else {
570                 temp2 = S.t - 0x80;
571                 temp2 = 0xC0 - temp2;

```

```

572     }
573     } /**/
574     MUL();
575     S.f += S.F;
576 }
577 #endif
578
579
580 #if defined(Guitar1) || defined(Guitar2)
581     if (S.bEnable && pin_ENABLE_INS) {
582
583         // READING SINUS
584         temp1 = guitar[*((char*)&S.f+1) & 0x3F];
585         // FORMING SIGNAL_GUITAR
586         if (S.t > 64) {
587             temp2 = S.t >> 2;
588             temp2 = 64 - temp2;
589         } else if (S.t >= 4) {
590             temp2 = 128 - S.t;
591         } else {
592             if (!S.t) {
593                 temp2 = prs << 2;
594             } else {
595                 temp2 = 255;
596                 if (S.t & 2) temp2 &= ~0x40;
597                 if (S.t & 1) temp2 &= ~0x20;
598             }
599         }
600         MUL();
601         S.f += S.F;
602     }
603 #endif
604
605
606
607 //-----
608
609     temp_dac >>= 2;
610     m_cDAC = *((char*)&temp_dac+0) + 0x80; // Next value to out
611
612     if (min > m_cDAC) min = m_cDAC;
613     if (max < m_cDAC) max = m_cDAC;
614
615 #if DAC_TYPE == DAC_TYPE_PWM
616     m_cDAC >>= 2;
617     CCP1CONbits.CCP1X = 0;
618     CCP1CONbits.CCP1Y = 0;
619     if (temp_dac & 2) CCP1CONbits.CCP1X = 1;
620     if (temp_dac & 1) CCP1CONbits.CCP1Y = 1 ;
621     CCP1L = m_cDAC;
622
623 #endif
624

```

```
625 //-----
626 prs++;
627 if (prs & 0x40) {
628     prs = 0;
629
630     if (S.t!=0xFF) S.t ++;
631
632
633     OS_Timer();
634 }
635 //-----
636
637
638 // LATAbits.LATA4 = 0;
639 }
```