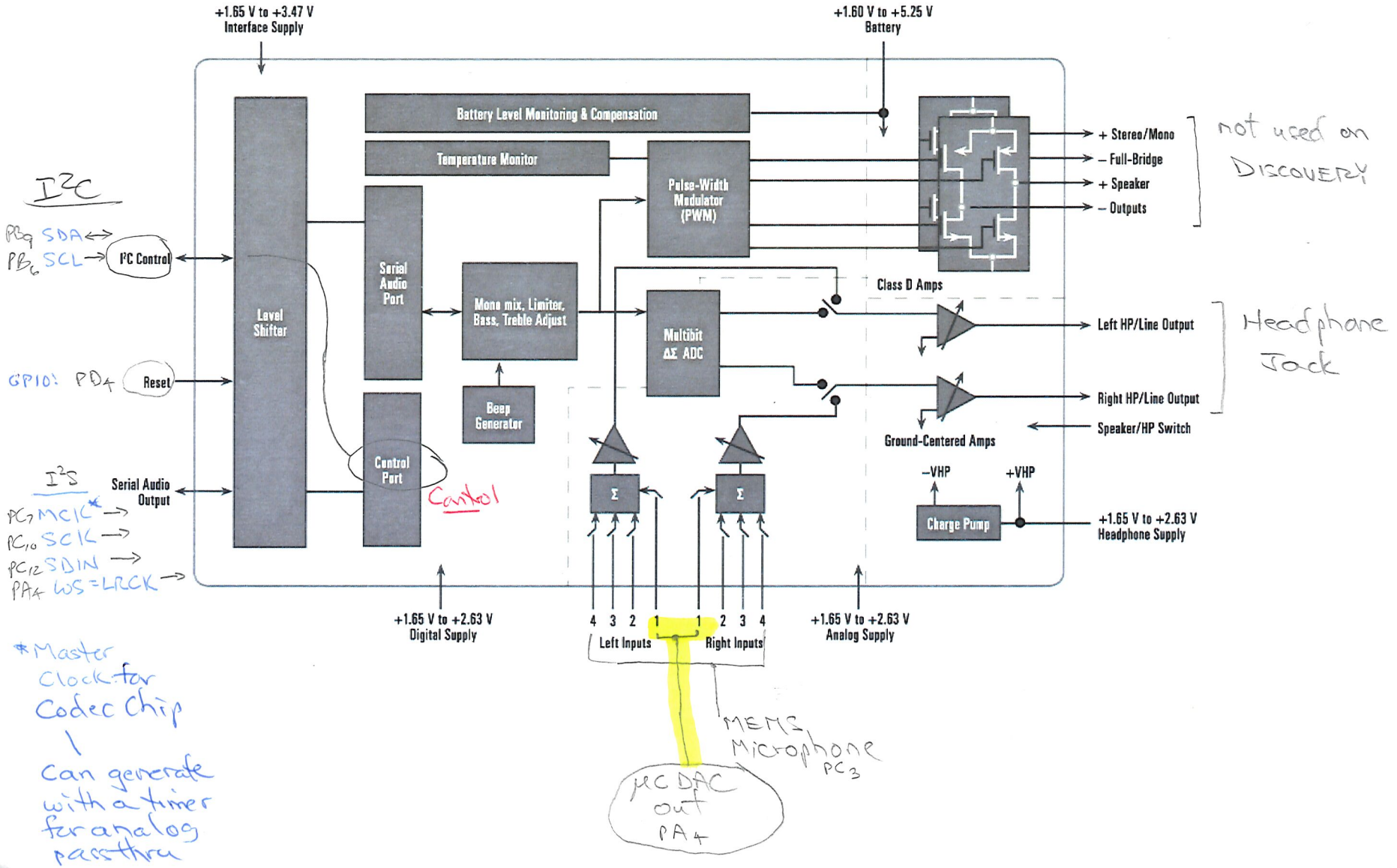
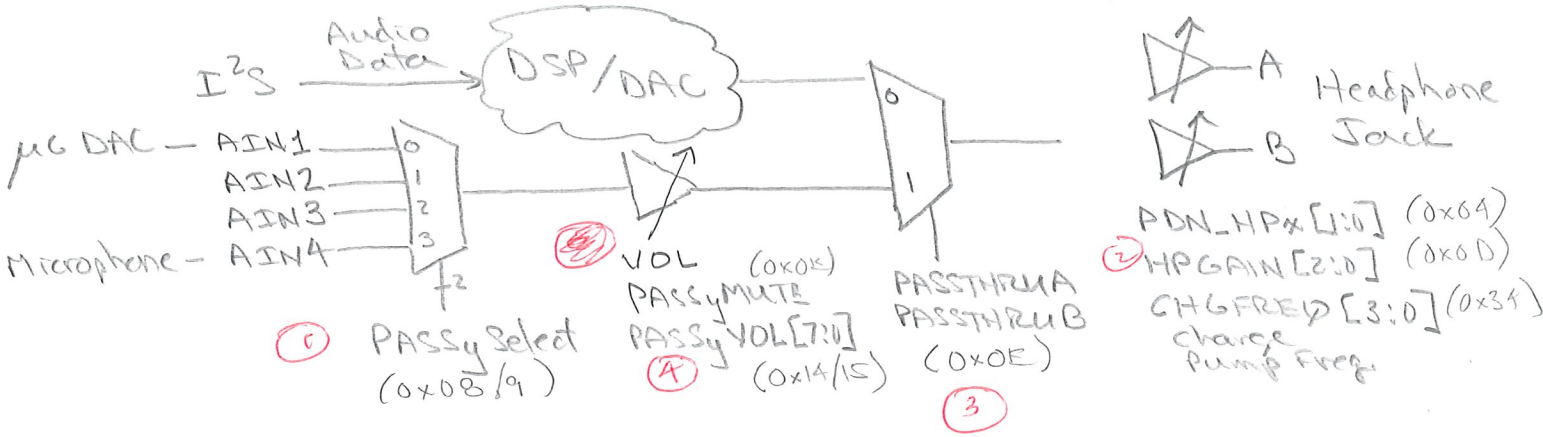
Cirrus Logic CS43L22 Portable Audio DAC with Integrated Class D Speaker Driver



Audio Codec Analog Passthru



Codec Registers

SW INIT

① 0x01 (0x02) - Power Control 1 — down = 0x01 (before configuring); up = 0x9E (to start audio)

⑧ 0x9E

② 0xAF (0x04) - Power Control 2

7	6	5	4	3	2	1	0
B	A	Spkr. Power					

10 = ON, 11 = OFF → PDN_HPA[1:0], PDN_HPB[1:0]

? 0x81 (0x05) - Clock Autodetect for CS43L22 speed

④ CODEC-STANDARD (0x06) I²S Interface Control (slave mode, Audio std. Philips fmt.)

⑤ 0x01 0x01 (0x08, 0x09) - Enable Passthru A/B (Select AIN channel)

3	2	1	0
AIN4	AIN3	AIN2	AIN1

AIN 1 2 3 4 → PASSYselect

① Playback Control 1

(0x0D) Headphone Gain / Playback Volume

HP GAIN2	1	0	A=B	B	A	B	A

HP gain; A=B; Master Mute; Invert PCM

② (0x0E) Misc. Controls

⑥ 0xC0

7	6	5	4				
B	A	B	A	Freeze	Deempl	Digsoft	Digrep

PASSTHRUA, PASSTHRUB: 1 = select analog

PASSAMUTE, PASSBMUTE: 1 = mute

③

⑦ 0x00 (0dB) (0x14, 0x15) Passthru x Volume - analog gain from -40db to 12dB in 0.5db steps

A B

④ PASSAVOL[7:0]

PASSBVOL[7:0]

(0x20, 0x21) Volume

( ) (0x34) Charge Pump Freq.

7	6	5	4				
N				-	-	-	-

$f = \frac{64 \cdot fs}{N+2}$ (N=5 by default)

0x0F (0x1F) Tone Control

7	6	5	4	3	2	1	0
Treble				Bass			

0x0A (0x1A, 0x1B) PCMx Volume - for SDIN pin to DSP; Master Volume - signal out of DSP

A B

cs43l22.c

only function used is "init".

```
/**
  * @brief Initializes the audio codec and the control interface.
  * @param DeviceAddr: Device address on communication Bus.
  * @param OutputDevice: can be OUTPUT_DEVICE_SPEAKER, OUTPUT_DEVICE_HEADPHONE,
  *                      OUTPUT_DEVICE_BOTH or OUTPUT_DEVICE_AUTO .
  * @param Volume: Initial volume level (from 0 (Mute) to 100 (Max))
  * @retval 0 if correct communication, else wrong communication
  */
#include "cs43l22.h"

uint32_t cs43l22_Init(uint16_t DeviceAddr, uint16_t OutputDevice, uint8_t Volume, uint32_t
AudioFreq)
{
  uint32_t counter = 0;

  /* Initialize the Control interface of the Audio Codec */
  AUDIO_IO_Init();

  /* Keep Codec powered OFF */
  counter += CODEC_IO_Write(DeviceAddr, CS43L22_REG_POWER_CTL1, 0x01);

  /*Save Output device for mute ON/OFF procedure*/
  switch (OutputDevice)
  {
  case OUTPUT_DEVICE_SPEAKER:
    OutputDev = 0xFA;
    break;

  case OUTPUT_DEVICE_HEADPHONE:
    OutputDev = 0xAF;
    break;

  case OUTPUT_DEVICE_BOTH:
    OutputDev = 0xAA;
    break;

  case OUTPUT_DEVICE_AUTO:
    OutputDev = 0x05;
    break;

  default:
    OutputDev = 0x05;
    break;
  }

  counter += CODEC_IO_Write(DeviceAddr, CS43L22_REG_POWER_CTL2, OutputDev);

  /* Clock configuration: Auto detection */
  counter += CODEC_IO_Write(DeviceAddr, CS43L22_REG_CLOCKING_CTL, 0x81);

  /* Set the Slave Mode and the audio Standard */
  counter += CODEC_IO_Write(DeviceAddr, CS43L22_REG_INTERFACE_CTL1, CODEC_STANDARD);
```

Call: AUDIO_I2C_ADDRESS | OUTPUT_DEVICE_HEADPHONE | 50 | 20000

0x02

Power Control 2 (Reg. 0x04)

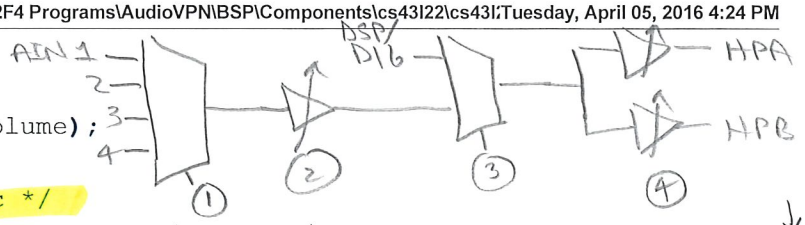
HPB	HPA	SPKB	SPKA
10	10	11	11
Headphone always on.		speakers always off	

0x04

0x05

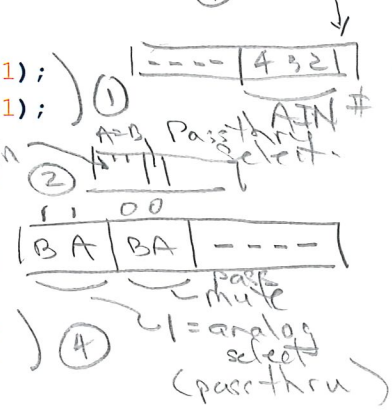
0x06

0x04 = Philips I²C Std

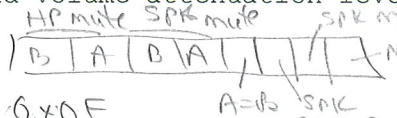



```
  /* Set the Master volume */ -0x20/21
  counter += cs43122_SetVolume(DeviceAddr, Volume);

/*** ANALOG PASSTHRU SETUP - NOT IN cs42122.c */
  /*** Select input AIN1 as PASSTHRU A and PASSTHRU B */ 0x08/9
  counter += CODEC_IO_Write(DeviceAddr, CS43L22_REG_PASSTHR_A_SELECT, 0x01);
  counter += CODEC_IO_Write(DeviceAddr, CS43L22_REG_PASSTHR_B_SELECT, 0x01);
  /*** Set analog amplifier gain */
  counter += CODEC_IO_Write(DeviceAddr, CS43L22_REG_PLAYBACK_CTL1, 0x70); 0x0D Headphone Gain
  /*** Select PASSTHRU, rather than DSP/DAC output */
  counter += CODEC_IO_Write(DeviceAddr, CS43L22_REG_MISC_CTL, 0xC0); 0x0E
  /*** Set headphone amplifier gain for A and B */
  counter += CODEC_IO_Write(DeviceAddr, CS43L22_REG_PASSTHR_A_VOL, 0x05); 0x14/15
  counter += CODEC_IO_Write(DeviceAddr, CS43L22_REG_PASSTHR_B_VOL, 0x05);
/***  END ANALOG PASSTHRU SETUP  ***/
```




```
  /* If the Speaker is enabled, set the Mono mode and volume attenuation level */
  if(OutputDevice != OUTPUT_DEVICE_HEADPHONE)
  {
    /* Set the Speaker Mono mode */
    counter += CODEC_IO_Write(DeviceAddr, CS43L22_REG_PLAYBACK_CTL2, 0x06); 0x0F
    /* Set the Speaker attenuation level */
    counter += CODEC_IO_Write(DeviceAddr, CS43L22_REG_SPEAKER_A_VOL, 0x00); 0x24/25
    counter += CODEC_IO_Write(DeviceAddr, CS43L22_REG_SPEAKER_B_VOL, 0x00);
  }
```

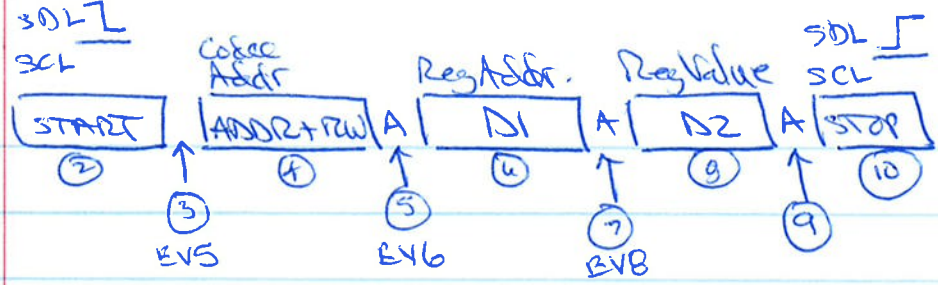



```
  /* Additional configuration for the CODEC. These configurations are done to reduce
  the time needed for the Codec to power off. If these configurations are removed,
  then a long delay should be added between powering off the Codec and switching
  off the I2S peripheral MCLK clock (which is the operating clock for Codec).
  If this delay is not inserted, then the codec will not shut down properly and
  it results in high noise after shut down. */

  /* Disable the analog soft ramp */
  counter += CODEC_IO_Write(DeviceAddr, CS43L22_REG_ANALOG_ZC_SR_SETT, 0x00); 0x0A
  /* Disable the digital soft ramp */
  //counter += CODEC_IO_Write(DeviceAddr, CS43L22_REG_MISC_CTL, 0x04); 0x0E
  /* Disable the limiter attack level */
  counter += CODEC_IO_Write(DeviceAddr, CS43L22_REG_LIMIT_CTL1, 0x00); 0x27
  /* Adjust Bass and Treble levels */
  counter += CODEC_IO_Write(DeviceAddr, CS43L22_REG_TONE_CTL, 0x0F); 0x1F
  /* Adjust PCM volume level */
  counter += CODEC_IO_Write(DeviceAddr, CS43L22_REG_PCMA_VOL, 0x0A); 0x1A/1B
  counter += CODEC_IO_Write(DeviceAddr, CS43L22_REG_PCMB_VOL, 0x0A);
```


```
/*** ADDED TO BYPASS "PLAY" Function: Power on the Codec */
  counter += CODEC_IO_Write(DeviceAddr, CS43L22_REG_POWER_CTL1, 0x9E);

  /* Return communication control value */
  return counter;
```

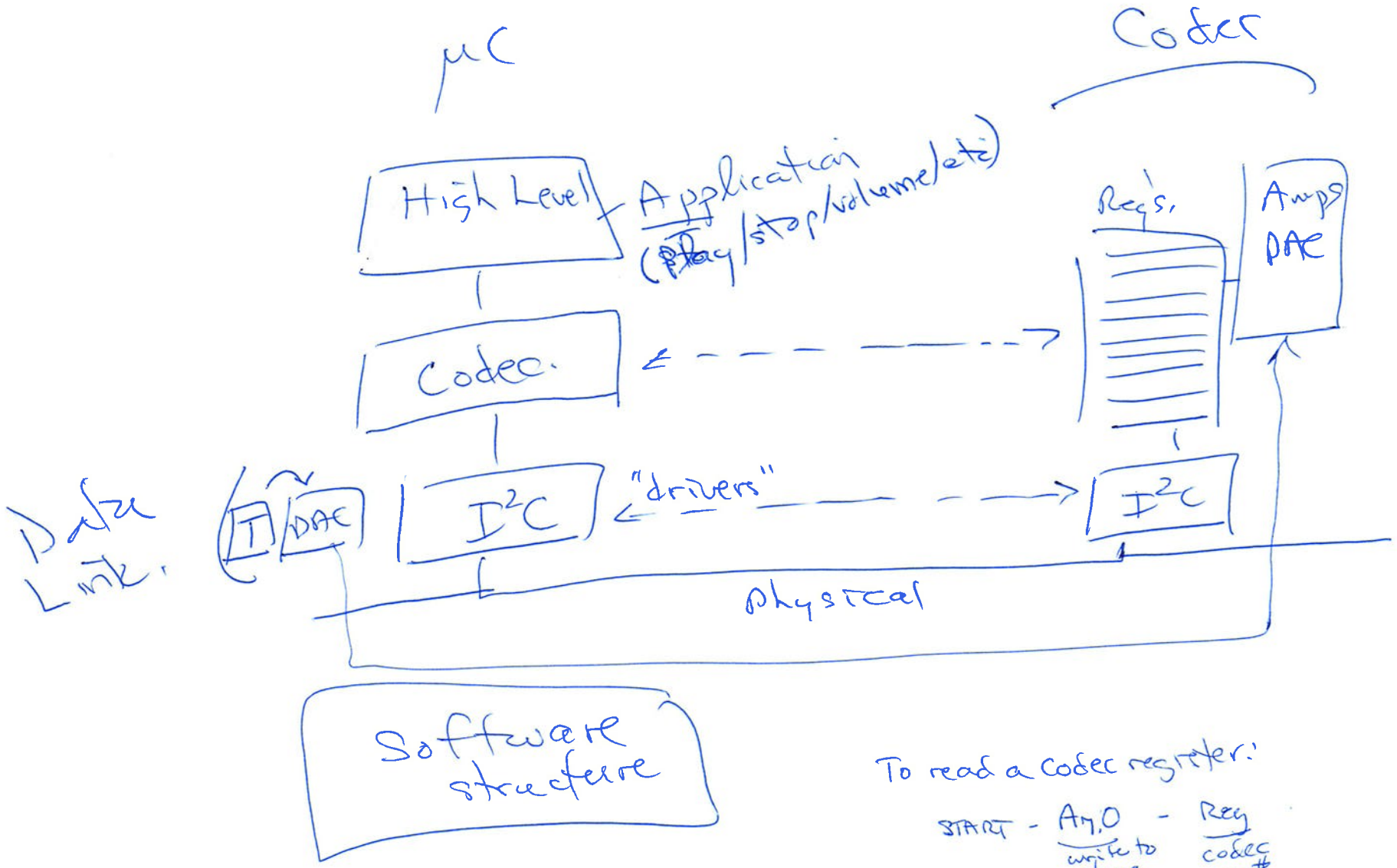


Codec_WriteRegister (RegisterAddr, RegisterValue)

1. • I2C_GetFlagStatus(·, I2C_FLAG_BUSY) - wait until bus not busy
2. • I2C_GenerateStart(·, ENABLE)
3. • I2C_CheckEvent (·, I2C_EVENT_MASTER_MODE_SELECTED)
   - we are now Master: EV5 => BUSY, MSL, SB flags. (start generated)
4. I2C_Send7bitAddress(·, CODEC_ADDRESS, I2C_Direction_Transmit) — 1001 01A0, R/W = 0
5. I2C_CheckEvent(·, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECT)
   EV6: ACK received from slave - we can xmit as master.
   Flags: BUSY, MSL, ADDR, TXE, TRA (end addr. xmit; xmit buffer empty; R/W xmit)
6. I2C_SendData(·, RegisterAddr) - Codec reg. #
7. I2C_CheckEvent(·, I2C_EVENT_MASTER_BYTE_TRANSMITTING)
   EV8: data now shifting out to SDL.
   Flags: BUSY, MSL, TXE, TRA
8. I2C_SendData(·, RegisterValue) - Codec reg. value
9. !I2C_GetFlagStatus(·, I2C_FLAG_BTF) (all data bytes finished)
10. I2C_GenerateStop(·, ENABLE)

EV1 - address match
EV2 - slave byte received
EV3 - slave byte sent
EV4 - slave STOP detected
EV7 - master byte received
EV9 - master 10-bit addr. mode

Test flags in SR1 : SR2



To read a codec register:

START - A7:0 (write to slave) - Reg (codec reg #)

START - A7:1 (read slave) . Value (reg. value from slave) STOP