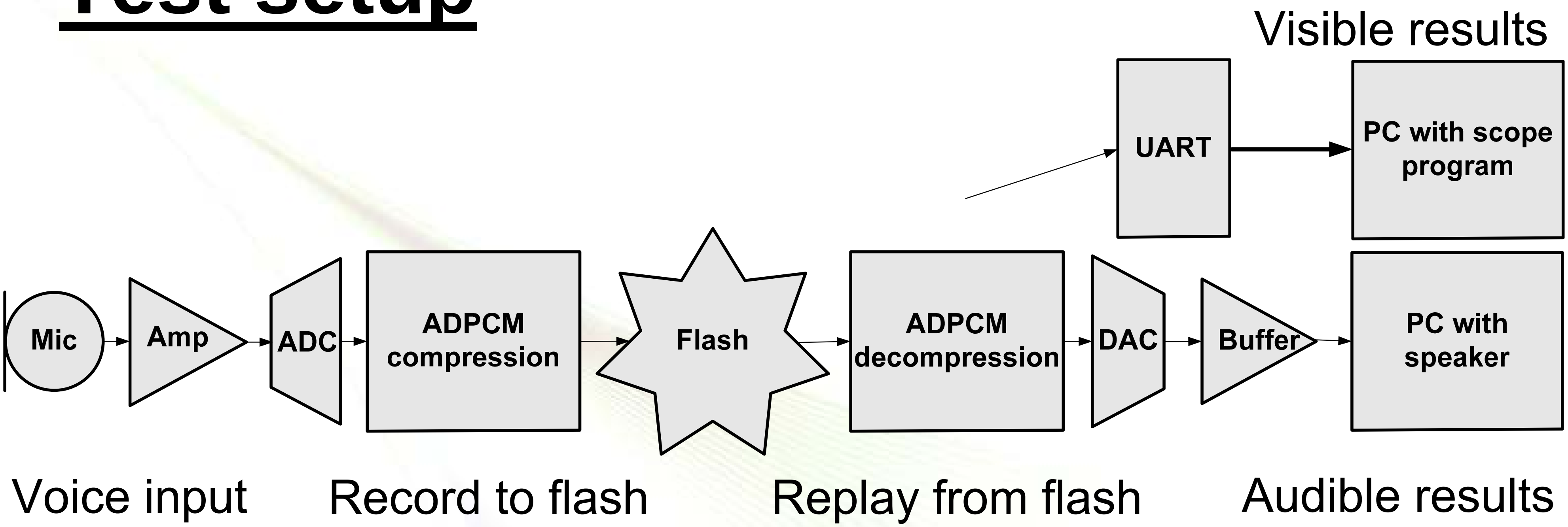
Hands-On: Realizing the MSP430 Signal Chain through ADPCM

Steve Underwood
MSP430 FAE Asia
Texas Instruments

An outline of this session

- Using the analog capabilities of the MSP430FG4619 to build a signal chain
- Getting a basic signal chain up & running
- Which speech codecs make sense for us?
- Code implementing ADPCM
- Experimenting with the application

Test setup

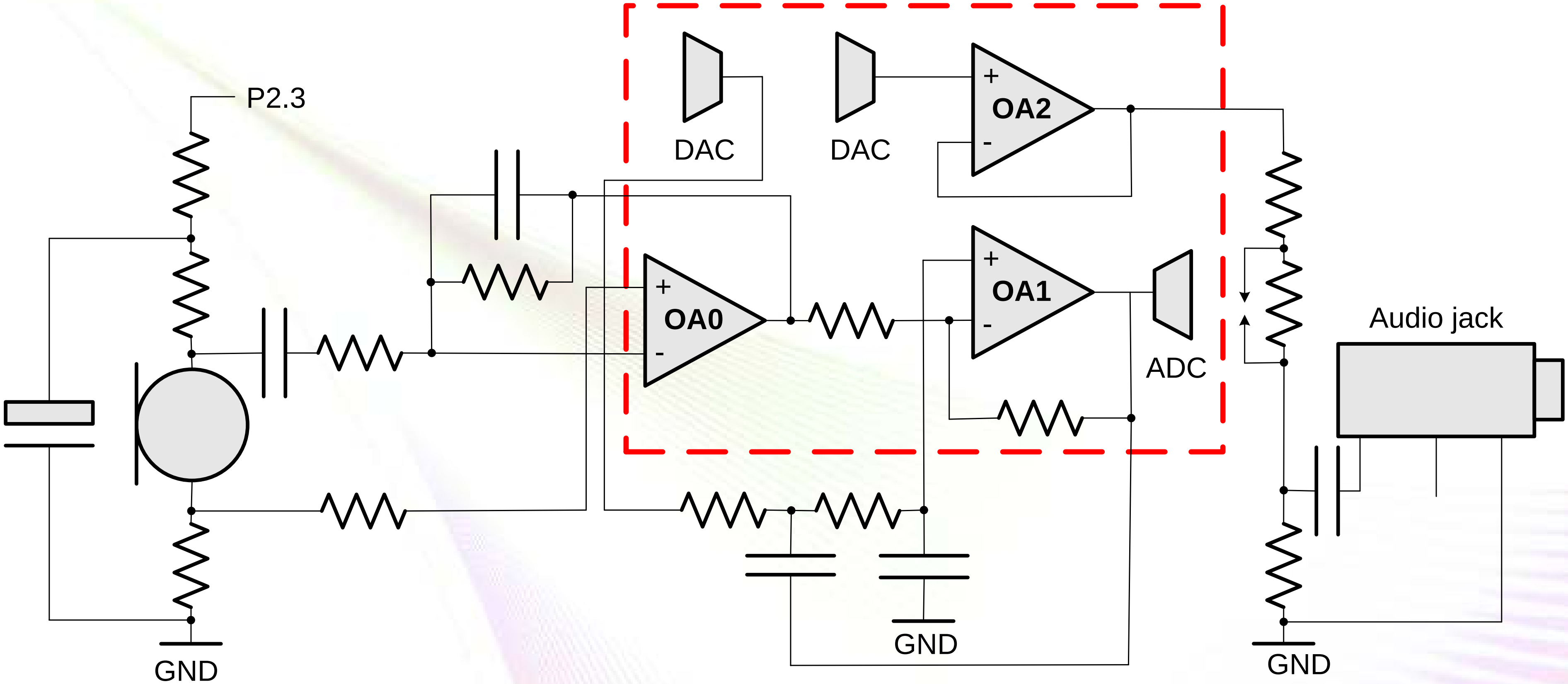


- A microphone amp, to capture your voice
- Mic > amp > ADC12 > compression > flash
- Flash > decompression > DAC12 > amp > PC
- Simple PC scope program to handle output

An outline of this session

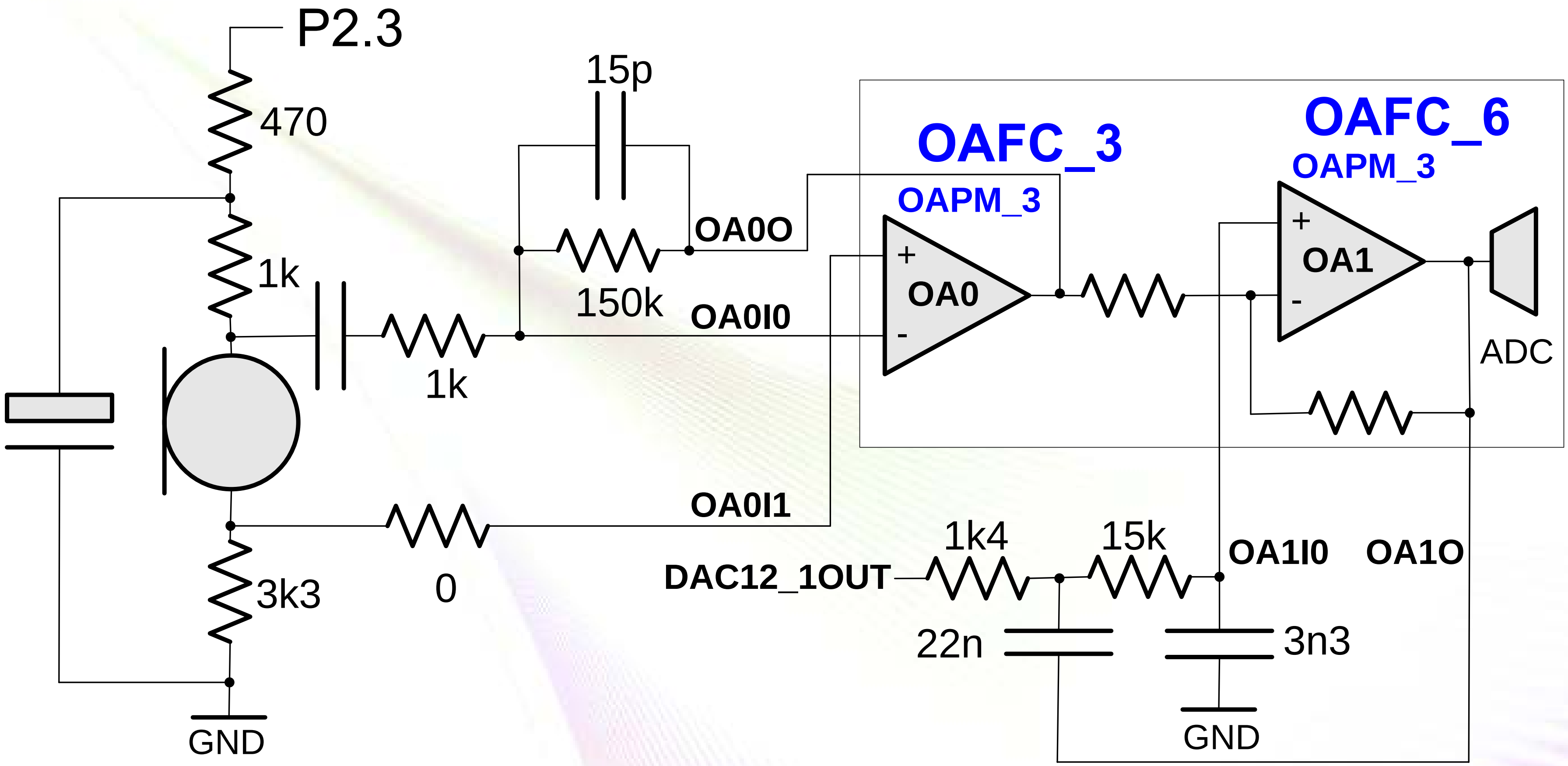
- Using the analog capabilities of the MSP430FG4619 to build a signal chain
- Getting a basic signal chain up & running
- Which speech codecs make sense for us?
- Code implementing ADPCM
- Experimenting with the application

External components



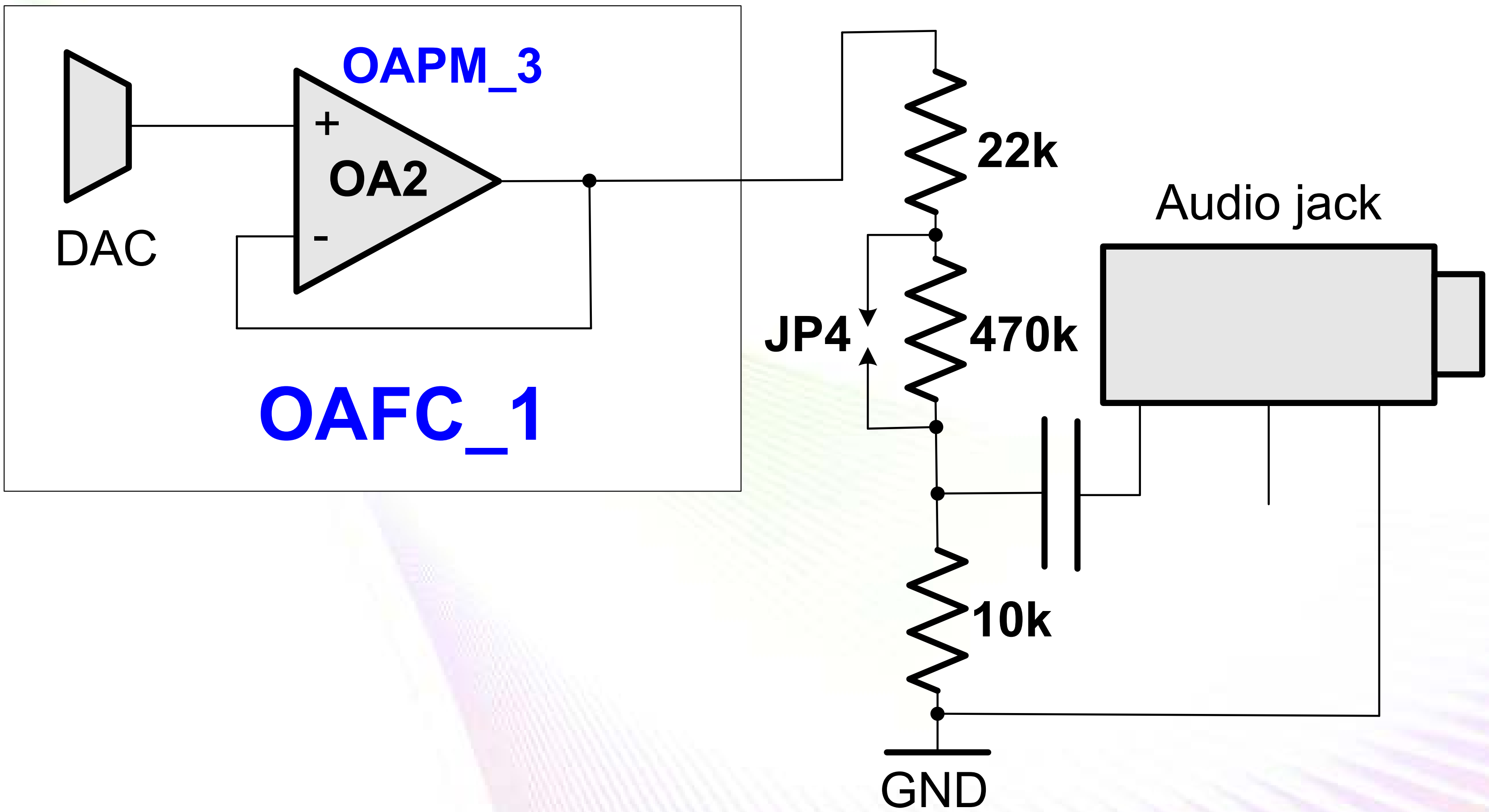
- **The red box shows MCU elements**
- **Very little external circuitry is required to make a real signal chain work**

Microphone amplifier configuration




```
OA0CTL1 = OAFC_0;                /* General purpose amp */
OA0CTL0 = OAN_0 | OAP_1 | OAPM_3;
OA1CTL1 = OAFC_6 | OAFBR_2;  /* Inverting PGA */
OA1CTL0 = OAN_2 | OAP_0 | OAPM_3 | OAADC1;
```

Output amplifier configuration




```
OA2CTL1 = OAFC_1;                /* Unity gain buffer */
OA0CTL0 = OAP_2 | OAPM_3 | OAADC1;
```

Let's get started…..

- **Find project "adpcm", open it, build it, load it into the MSP430FG4619 on your board, and run it**
- **Connect the ATC board's 3.5mm jack to your PC's audio input jack**
- **Find the program "scope.exe", and run it on your PC**
  - Click on the "Snd card" button to light the yellow mark
  - You should see the output from your mic
  - If you can't hear anything, click on "Snd pass"

What is the code doing?

- It sets up the analog circuitry
- It initializes timer A to produce a sampling “tick” 8003.4 times/second
- It digitizes the mic. signal every tick
- It uses a single pole LPF to estimate and remove DC

```
int32_t estimate;
estimate += ((((int32_t) signal << 16) - estimate) >> 10);
signal -= (*p >> 16);
```

- It converts the signal back to analog
- It sends the analog signal to your PC

An outline of this session

- Using the analog capabilities of the MSP430FG4619 to build a signal chain
- Getting a basic signal chain up & running
- Which speech codecs make sense for us?
- Code implementing ADPCM
- Experimenting with the application

Why ADPCM?

To suit our goals we need a speech codec with:

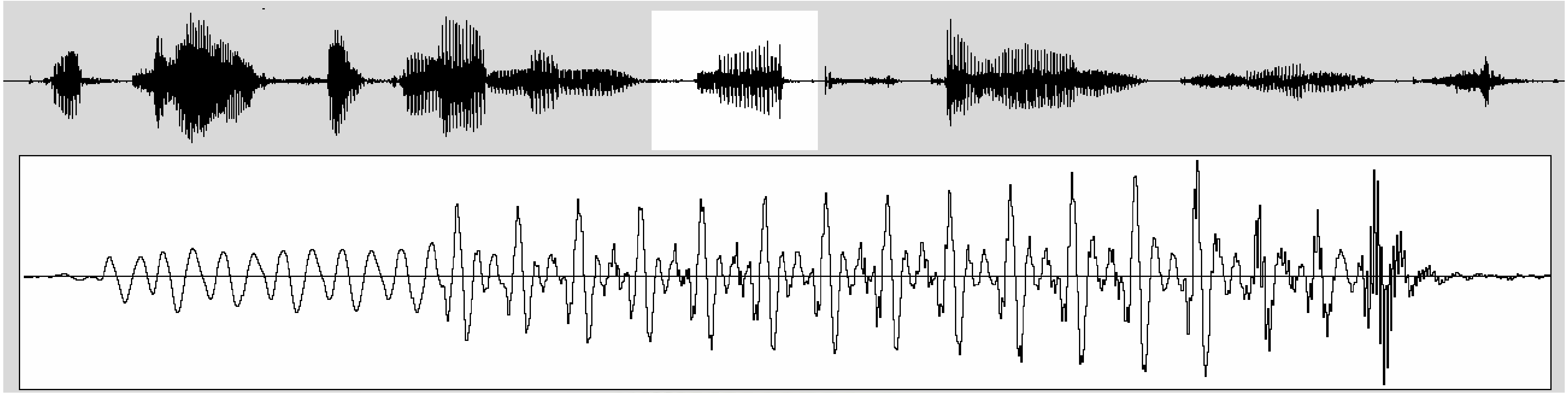
- **A fairly low bit rate**
- **Fairly low computational complexity**
- **Modest code and buffer requirements**

- **A-law/μ-law are simple, but the bit rate is high**
- **LPC can be complex to encode, less complex to decode. Might be good for alarms**
- **RPE, CELP, and other schemes offer good quality, but require a lot of computation and memory**
- **ADPCM offers a balance that fits our requirements**

What is ADPCM?...

- **The μ-law and A-law codecs, used for the PSTN, compress 12 bit (72dB dynamic range) audio to 8 bits in a pseudo-logarithmic manner**
  - They use 8 linear sections to approximate logarithmic
  - Give a fairly constant signal to distortion ratio (~30dB)
  - No state information carried from sample to sample
  - Encoded samples are a 3 bit “section” + a 4 bit linear value + a sign bit (c.f. characteristic + mantissa)

…What is ADPCM?...



- **What if we encode the difference between successive samples, rather than the samples themselves?**
  - In low frequency sections this lets us encode in finer steps
  - In very high frequency sections it can increase the coarseness
  - Overall, it encodes the great majority of encoded samples more accurately
  - Decoding should start from the same level as the encoder

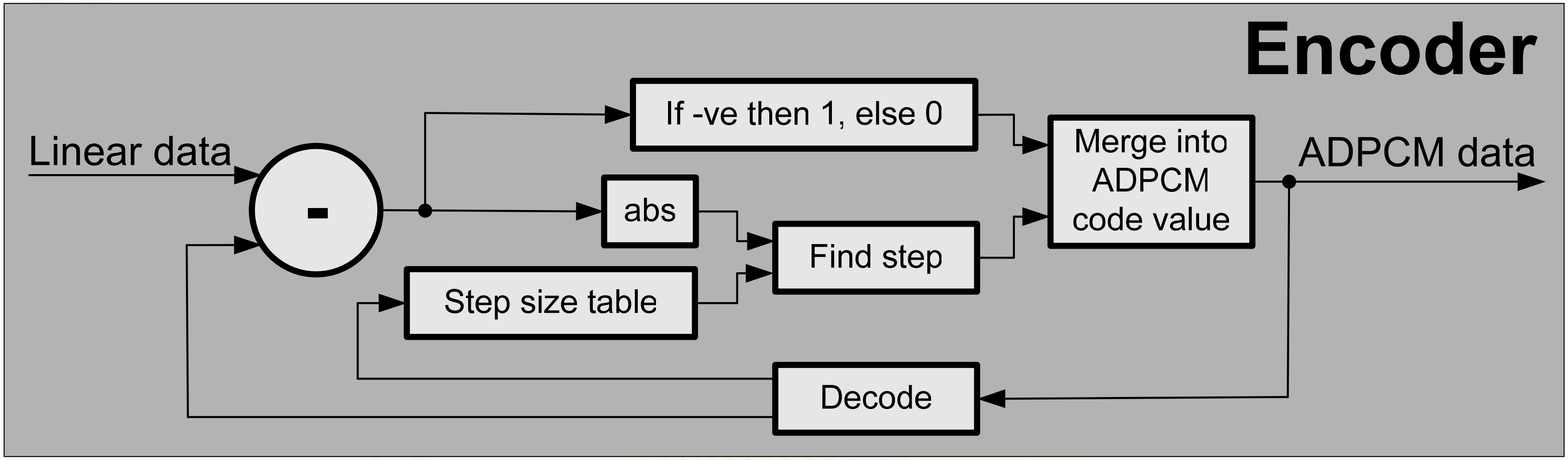
...What is ADPCM?...

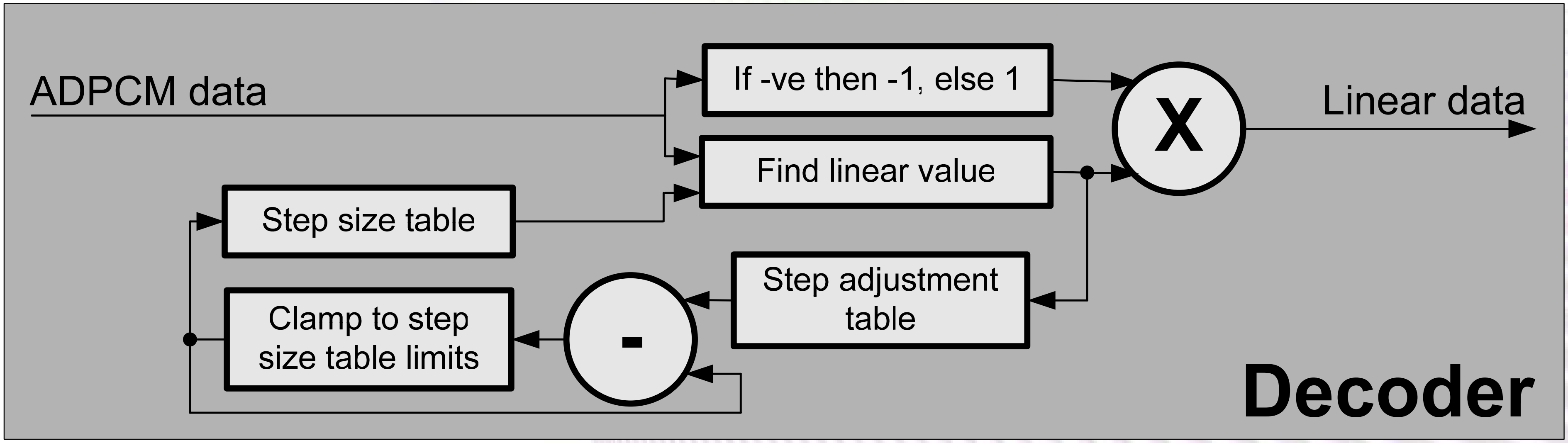
- **Speech energy varies fairly slowly**
- **If we track the short term level, can we use variable step sizes, but avoid encoding the step size itself?**
  - Could save 3 out of 8 bits by applying this idea directly to A-law or u-law
- **Key issue: we need to ensure the decoder will exactly track the encoder in choosing step sizes**
- **We send the quantized difference between the current sample, and the decoded version of the previous sample**
  - Both the encoder and decoder know these values, so they can track each other's step size choices
  - Means the decoder must be nested inside the encoder

…What is ADPCM?

- **The quantization step size adapts every sample, to follow the short term signal level.**
  - Adapting every sample is part of making the two ends track each other
  - If the quantized sample is small, we change to finer steps (< range)
  - If the quantized sample is large, we change to coarser steps (> range)
  - We prevent overrunning the ends of the step size table
  - The step pattern used is a key difference between ADPCM codecs
- **Well quantized 4 bit (sign + 3 bit level) differences work almost as well as μ-law/A-law**
  - Heavily used for speech storage, in things like voice mail systems
- **3 or 2 bit differences can give clear speech**
  - Fine for things like alarms

Putting it all together





What limitations does ADPCM have?

- **The decoder must track the encoder**
  - Any bit errors in the data which upset decoding
  - The decoder must start decoding from the start of the encoded sequence
- **This limitation can be mitigated by inserting periodic unencoded samples**
  - Decoding can pick up from any unencoded sample, as long as we know where to look for it
- **Tone bursts may suffer some corruption**
  - Affects things like signaling tones (e.g. DTMF) on the PSTN
  - Not usually important for speech storage

Which ADPCM?

- **ADPCM is a general class of audio compression**
- **ITU G.726 is widely used for PSTN calls**
  - Variants from 16kbps to 40kbps
  - Has a lot of complexity to achieve things we don't care about
- **OKI ADPCM is widely used for IVR and voice mail**
  - Good quality; low compute requirements; low memory requirements
  - 32kbps. Sometimes reduced to 24kbps. No lower bit rate option
- **IMA (DVI, Intel) ADPCM is similar to OKI ADPCM**
  - Good quality; low compute requirements; low memory requirements
  - 32kbps
- **Other ADPCM algorithms, at 2, 3 or 4 bits per sample**
  - At 6k samples/second, 2 bit ADPCM runs at 12kbps
  - Low bit rate ADPCM and a PWM D/A converter can add voice alerts to small MSP430 parts

An outline of this session

- Using the analog capabilities of the MSP430FG4619 to build a signal chain
- Getting a basic signal chain up & running
- Which speech codecs make sense for us?
- Code implementing ADPCM
- Experimenting with the application

IMA ADPCM state information

```
typedef struct
{
    int last;
    int step_index;
} ima_adpcm_state_t;
```

```
static const int step_adjustment[8] =
{
    -1, -1, -1, -1, 2, 4, 6, 8
};
```

IMA ADPCM step table

```
static const int step_size[89] =
{
        7,      8,      9,     10,     11,     12,     13,     14,
       16,     17,     19,     21,     23,     25,     28,     31,
       34,     37,     41,     45,     50,     55,     60,     66,
       73,     80,     88,     97,    107,    118,    130,    143,
      157,    173,    190,    209,    230,    253,    279,    307,
      337,    371,    408,    449,    494,    544,    598,    658,
      724,    796,    876,    963,   1060,   1166,   1282,   1411,
     1552,   1707,   1878,   2066,   2272,   2499,   2749,   3024,
     3327,   3660,   4026,   4428,   4871,   5358,   5894,   6484,
     7132,   7845,   8630,   9493,  10442,  11487,  12635,  13899,
    15289,  16818,  18500,  20350,  22385,  24623,  27086,  29794,
    32767
};
```

IMA ADPCM encode

```
uint8_t ima_adpcm_encode(ima_adpcm_state_t *s, int16_t linear)
{
    int e, ss, adpcm, diff, initial_e;
    ss = step_size[s->step_index];
    initial_e = e = linear - s->last;
    diff = ss >> 3;         adpcm =  (uint8_t) 0x00;
              if (e < 0)    {adpcm =  (uint8_t) 0x08; e  = -e;}
    ss >>= 1; if (e >= ss) {adpcm |= (uint8_t) 0x04; e -= ss;}
    ss >>= 1; if (e >= ss) {adpcm |= (uint8_t) 0x02; e -= ss;}
              if (e >= ss) {adpcm |= (uint8_t) 0x01; e -= ss;}
    diff = (initial_e < 0)
        ?  initial_e + e - diff :  initial_e - e + diff;
    s->last = diff + s->last;
    s->step_index += step_adjustment[adpcm & 0x07];
    if (s->step_index < 0)       s->step_index = 0;
    else if (s->step_index > 88) s->step_index = 88;
    return adpcm;
}
```

IMA ADPCM decode

```
int16_t ima_adpcm_decode(ima_adpcm_state_t *s, uint8_t adpcm)
{
    int e, ss; int16_t linear;
    ss = step_size[s->step_index];
    e = ss >> 3;
    if (adpcm & 0x01) e += (ss >> 2);
    if (adpcm & 0x02) e += (ss >> 1);
    if (adpcm & 0x04) e += ss;
    if (adpcm & 0x08) e = -e;
    linear = s->last + e;
    s->last = linear;
    s->step_index += step_adjustment[adpcm & 0x07];
    if (s->step_index < 0)          s->step_index = 0;
    else if (s->step_index > 88)  s->step_index = 88;
    return linear;
}
```

An outline of this session

- Using the analog capabilities of the MSP430FG4619 to build a signal chain
- Getting a basic signal chain up & running
- Which speech codecs make sense for us?
- Code implementing ADPCM
- Experimenting with the application

Let’s compress and decompress

- **The lab software let’s us choose amongst**
  - IMA ADPCM (define “USE_IMA”)
  - Oki ADPCM (define “USE_OKI”)
  - Low bit rate ADPCM, suitable for things like spoken alarms (define “USE_2BIT”)
- **Make sure one of these is defined near the top of adpcm_app.c**
- **Adjust the ADC12 interrupt routine to encode and decode the digitized samples**
- **Experiment with quality of the various codecs**
  - IMA and Oki should be similar, and good
  - The 2-bit codec is poorer, but uses half the bit rate
- **Try reducing the sampling rate, to reduce the bit rate, if you have time**

Using extended flash memory in C

- **The IAR and CCE compilers support extended function pointers, but not extended data pointers**
  - Using extended memory for data storage requires special handling.
  - IAR provide the functions below, to allow direct reading or writing of any address in memory
- **We will use the direct memory access functions to store and retrieve audio from the upper 64k of flash memory**

```
int __data20_read_short(long int flash_addr);
void __data20_write_short(long int flash_addr, int value);
```

Erasing the flash memory

- **The CPU can easily erase flash memory**
  - Set up the flash timing generator
  - Set the flash control registers to enable erasing of the flash
  - Write any value into any location in the page to be erased
  - Wait for completion – only needed if not running from the Flash being erased (e.g. running code in RAM)
  - Set the flash control registers to prevent accidental writing to flash
  - All timing is handled by the hardware – about 16ms per page

```
FCTL2 = FWKEY | FSSEL_1 | FTG_???;
FCTL3 = FWKEY;                  /* Unlock the flash */
FCTL1 = FWKEY | ERASE;          /* Enable erasing */
__data20_write_short(ptr, 0);   /* Erase the flash page */
while (FCTL3 & BUSY);           /* Wait for completion */
FCTL1 = FWKEY;                  /* Disable erase/writing */
FCTL3 = FWKEY | LOCK;           /* Lock the flash */
```

Writing audio to the flash memory.

- **The CPU can easily write to flash memory**
  - Set up the flash timing generator.
  - Set the flash control registers to enable writing to the flash.
  - Write values into the required memory locations.
  - Wait for completion – only needed if not running from the Flash being erased (e.g. running code in RAM)
  - Set the flash control registers to prevent accidental writing to flash.
  - All timing is handled by the hardware – about 75us per word.
  - Writing is fast enough to keep up at 8000 samples/second.

```
FCTL2 = FWKEY | FSSEL_1 | FN_???;
FCTL3 = FWKEY;                   /* Unlock the flash */
FCTL1 = FWKEY | WRT;             /* Enable writing */
__data20_write_short(ptr, val); /* Write to the flash */
while (FCTL3 & BUSY);            /* Wait for completion */
FCTL1 = FWKEY;                   /* Disable erasing/writing */
FCTL3 = FWKEY | LOCK;            /* Lock the flash */
```

Let’s store and replay

- **The supplied code contains all the elements of a record and replay scheme**
- **Adjust the code to make those elements perform a recording to flash memory, followed by a repeating replay from flash memory**
  - LED4 will be on while record is in progress
  - The code contains the ability to restart recording during playback, by pressing the button in the corner of the board
- **Try the different codecs**
- **Try reducing the sampling rate, to see the effect on quality, and the minimum bit rate that might suit your needs**

Summary

- **The analog facilities in a number of MSP430 family devices are sufficient to realize complete practical signal chains with just a few passive components**
- **The processing capabilities of the MSP430 are sufficient to implement some interesting real world signal processing tasks**
- **What if I use an MSP430 with limited resources?**
  - Any of the MSP430 ADC converters are adequate for simple voice applications – ADC10, ADC12, SD16, SD16A, or even a slope converter built with just a comparator
  - Timers A and B have up/down modes that can be used to build an adequate PWM based DAC for many uses – e.g. voice alert replay of stored data
- **SPI interfaced flash memories permit large scale data logging at very low power with the MSP430**

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Telephony	www.ti.com/telephony
Low Power Wireless	www.ti.com/lpw	Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265