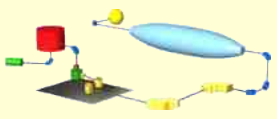


Università degli Studi di Catania
Facoltà di Ingegneria

Istituto Nazionale di Fisica Nucleare
Laboratorio Nazionale del Sud

Corso di laurea in Ingegneria Informatica

Gestione software di un sistema DAC / ADC ed organizzazione hardware

Candidato
Fortunato Oliveri

Relatore
Prof. Ing. Michele Malgeri

Correlatore
Dott. Paolo Finocchiaro

Anno accademico 2011/2012

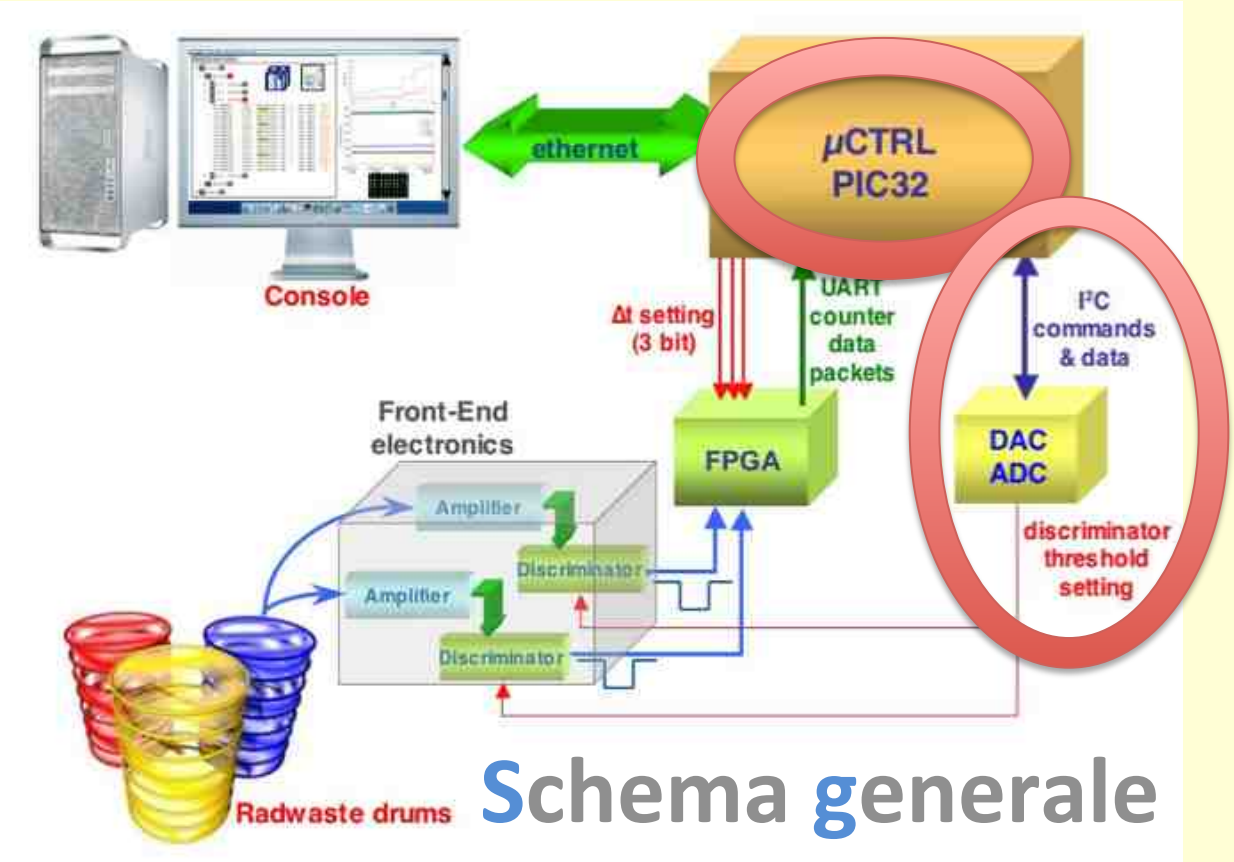
Indice

- Descrizione DMNR, punti salienti e motivazione scelte intraprese.
- Obiettivo del mio lavoro di tesi.
- Funzioni implementate.
- Sviluppi futuri, stack TCP/IP.

DMNR

Detector Mesh for Nuclear Repository

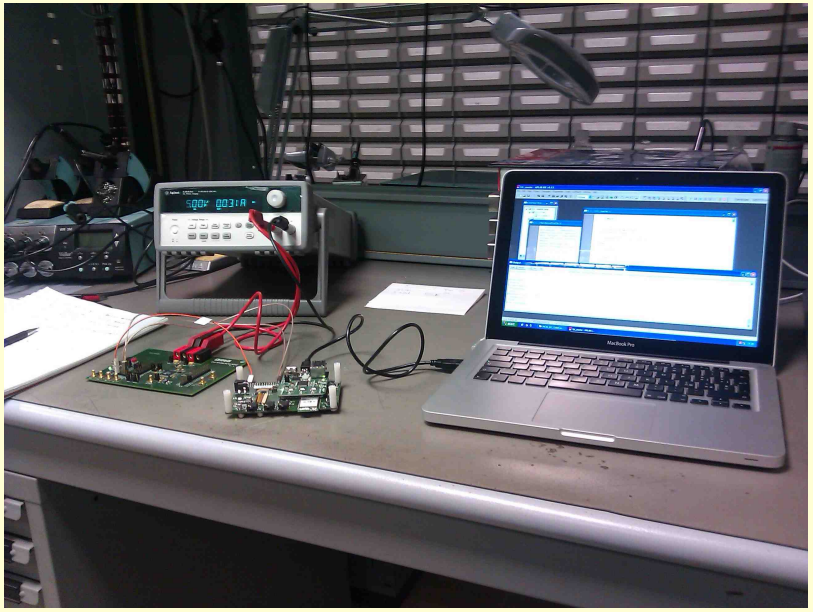
- Sistema per rivelazione di radiazioni real-time per depositi di scorie nucleari.
- Utilizzo di un microcontrollore PIC32 vista la versatilità e la potenza del dispositivo.
- Serve impostare un livello di soglia sui discriminatori, per modulare la generazione degli impulsi da contare.

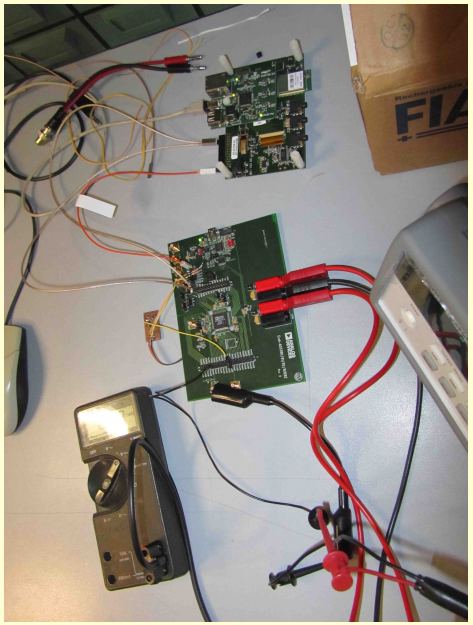


Schema generale

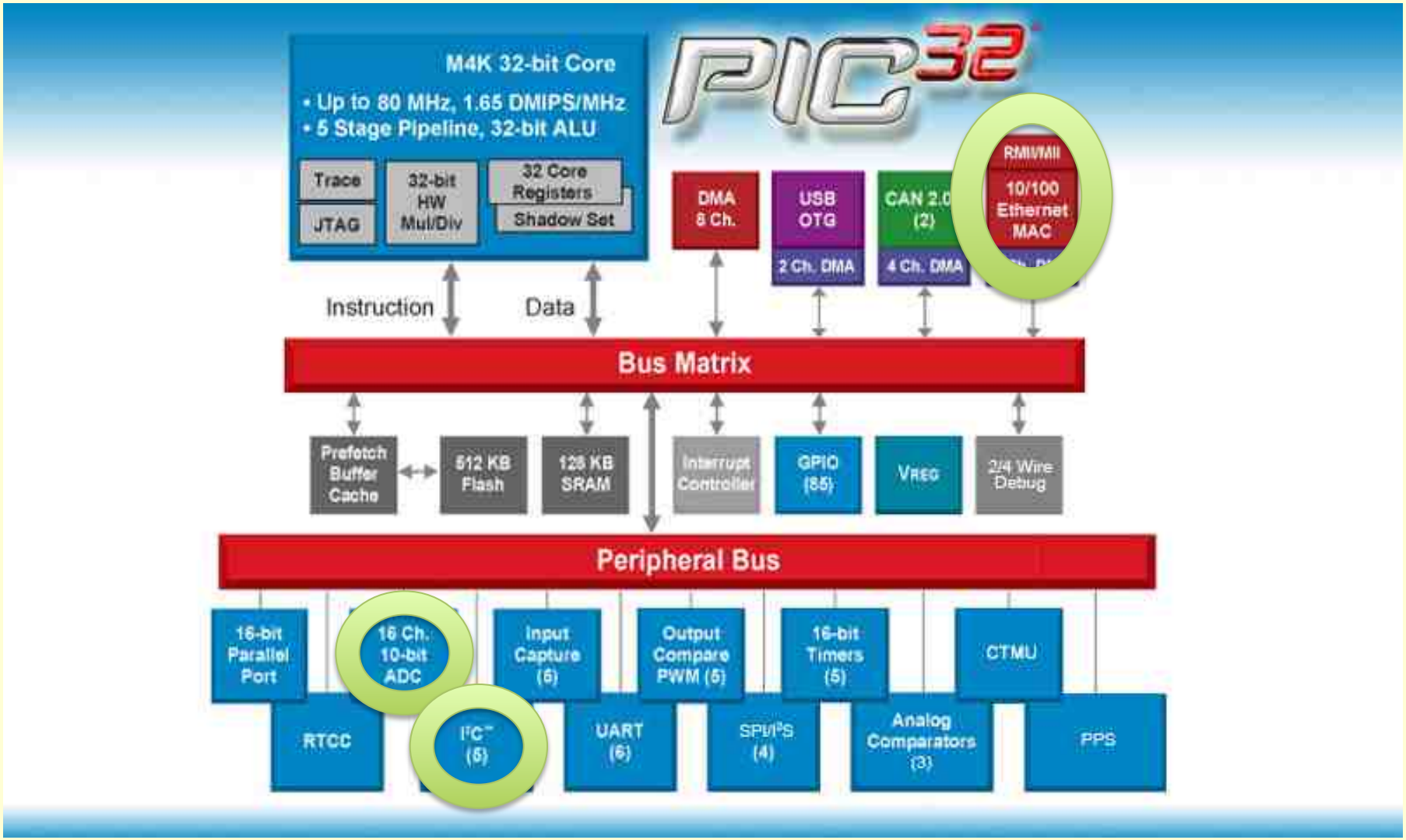
Obiettivi

- Sviluppare un software in grado di controllare a pieno un sistema DAC / ADC per la regolazione e la lettura di valori di tensione, essenziali per il funzionamento dei discriminatori.
- Organizzare correttamente l'hardware, per rendere il sistema equilibrato e funzionale.



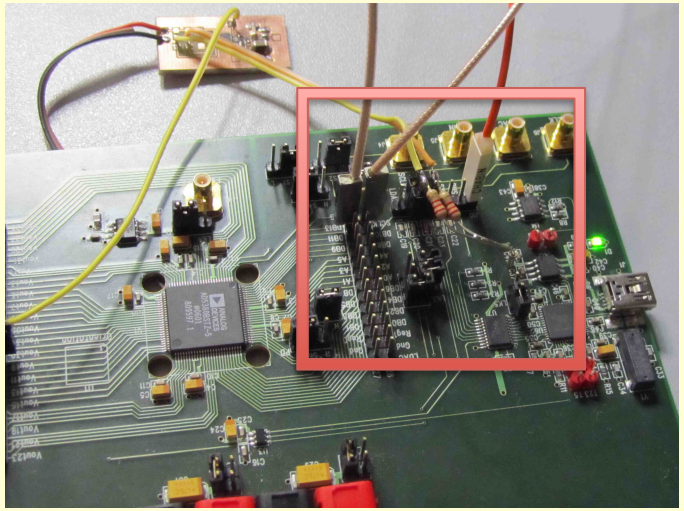


Gestione software di un sistema DAC / ADC ed organizzazione hardware



Il bus I^2C

- È stato scelto per la comunicazione tra microcontrollore e DAC.
- È un sistema di comunicazione seriale, basato su due linee SDA ed SCL.
- Funziona con un architettura master/slave.
- Non necessita di particolare circuiteria, sono fondamentali però le resistenze di pull-up.
- È un bus lento, anche se continuano ed essere standardizzate versioni sempre più veloci.



Principali funzioni implementate

```
UINT8 Calc_DacDataFormat(BOOL mostSB, float tensione){

	int risultato, cont, i;
	double pot, pot1;
	UINT8 msb, lsb;

	pot = pow(2, DAC_RESOLUTION);
	pot1 = pow(2, DAC_RESOLUTION - 1);

	risultato = (float) ( tensione * ( pot / ( 2 * REFERENCE_VOLTAGE_DAC ) ) * ( pot / ( GAIN_COEFFICIENT + 2 ) ) ) -
( OFFSET_COEFFICIENT - pot1 ) * ( 2 * REFERENCE_VOLTAGE_DAC ) / pot );

	msb = 0b00000000;
	lsb = 0b00000000;
	cont = DAC_RESOLUTION - 1;

	for(i = DAC_RESOLUTION; i > 0; i--){
		if( risultato >= pow(2, cont) ){
			if( pow(2, cont) > 128 ){
				msb = msb + pow(2, cont - 8);
			}else{
				lsb = lsb + pow(2, cont);
			}
			risultato = risultato - pow(2, cont);
			cont--;
		}else{
			cont--;
		}
	}
	if(mostSB){
		return msb;
	}else{
		return lsb;
	}
}
```

Funzione

Calc_DacDataFormat

Funzione reset

```
BOOL reset () {
// Initialize the data buffer
       I2C_FORMAT_7_BIT_ADDRESS(SlaveAddress, DAC_ADDRESS, I2C_WRITE);
//Scrivo il Controll Register
       i2cData[0] = SlaveAddress.byte;          // Address 1010101
       i2cData[1] = 0b00001100;                 // Pointer byte A5 to A0
       i2cData[2] = 0b00110110;                 // Data MSB
       i2cData[3] = 0b00000000;                 // Data LSB
// Reset di tutti i canali
       i2cData[4] = 0b00000010;                 // Pointer byte A5 to A0
       i2cData[5] = 0b00000000;                 // Data MSB
       i2cData[6] = 0b00000000;                 // Data LSB
       DataSz = 7;
       My_Init();
       My_StartTransfer();
       Success = My_TransmitOneByte(Success, DataSz, i2cData);
       StopTransfer();

       if(Success){
              return TRUE;
       } else {
              return FALSE;
       }
}
```

Funzione sveglia I2C

```
void svegliaI2C () {
// Initialize the data buffer
       I2C_FORMAT_7_BIT_ADDRESS(SlaveAddress, DAC_ADDRESS, I2C_WRITE);
//Scrivo il Controll Register
       i2cData[0] = SlaveAddress.byte;          // Address 1010101
       i2cData[1] = 0b00000000;                 // Pointer byte A5 to A0
       i2cData[2] = 0b00000000;                 // Data MSB
       i2cData[3] = 0b00000000;                 // Data LSB
// Reset di tutti i canali
       i2cData[4] = 0b00000000;                 // Pointer byte A5 to A0
       i2cData[5] = 0b00000000;                 // Data MSB
       i2cData[6] = 0b00000000;                 // Data LSB
       DataSz = 7;
       My_Init();
       My_StartTransfer();
       Success = My_TransmitOneByte(Success, DataSz, i2cData);
       StopTransfer();
}
```

Funzione SetThreshold

È la funzione più importante, perché permettere di impostare un valore di tensione su un qualsiasi canale del DAC, integrando la conversione del dato e il protocollo di invio.

```
BOOL SetThreshold (UINT canale, float tensione) {

// Initialize the data buffer
       I2C_FORMAT_7_BIT_ADDRESS(SlaveAddress, DAC_ADDRESS, I2C_WRITE);
//Scrivo il Controll Register
       i2cData[0] = SlaveAddress.byte;          // Address 1010101
       i2cData[1] = 0b00001100;                 // Pointer byte A5 to A0
       i2cData[2] = 0b00110110;                 // Data MSB
       i2cData[3] = 0b00000000;                 // Data LSB

// Imposto tensione canale
       i2cData[4] = canale;                                          // Pointer byte A5 to A0
       i2cData[5] = Calc_DacDataFormat(1, tensione) + 0b11000000;    // Data MSB
       i2cData[6] = Calc_DacDataFormat(0, tensione);                 // Data LSB

       DataSz = 7;

       My_Init();

       My_StartTransfer();
       Success = My_TransmitOneByte(Success, DataSz, i2cData);
       StopTransfer();

       if(Success){
              return TRUE;
       } else {
              return FALSE;
       }

}
```

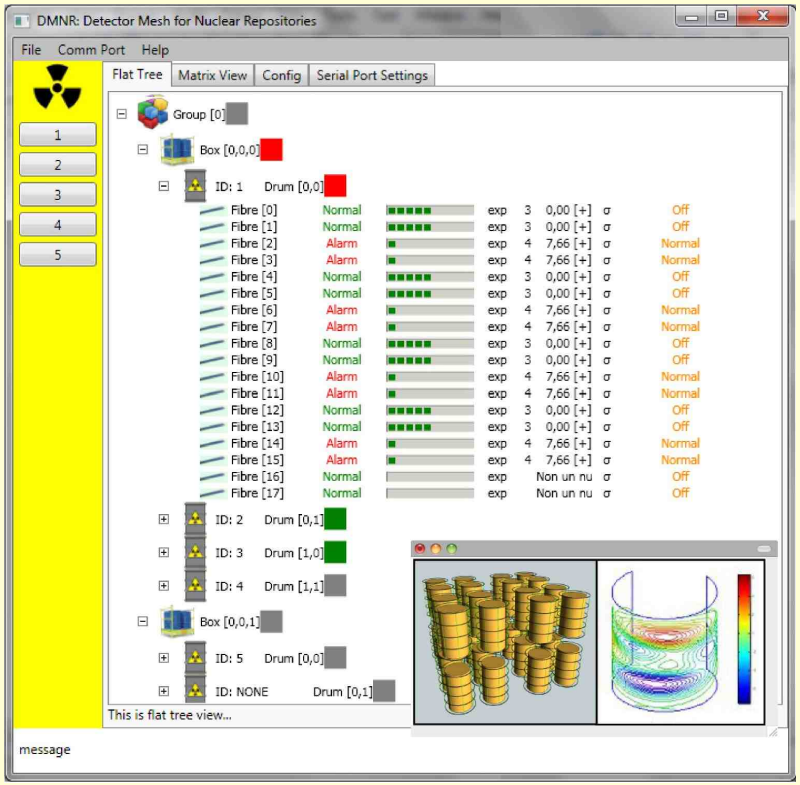
```
float ReadThreshold (UINT canale) {

/* Initialize the data buffer
        I2C_FORMAT_7_BIT_ADDRESS(SlaveAddress, DAC_ADDRESS, I2C_WRITE);
//Scrivo il Controll Register
        i2cData[0] = SlaveAddress.byte;             // Address 1010101
        i2cData[1] = 0b00001100;                    // Pointer byte A5 to A0
        i2cData[2] = 0b00110110;                    // Data MSB
        i2cData[3] = 0b00000000;                    // Data LSB
/* Settaggio monitor per il canale 0
        i2cData[4] = 0b00001010;                    // Pointer byte A5 to A0
        i2cData[5] = canale;                        // Data MSB
        i2cData[6] = 0b00000000;                    // Data LSB
        DataSz = 7;
        My_Init();
        My_StartTransfer();
        Success = My_TransmitOneByte(Success, DataSz, i2cData);
        StopTransfer();

int i;          //NOP
for(i = 10000; i > 0; i-- ){}

        // configure and enable the ADC
        CloseADC10();
        #define PARAM1  ADC_FORMAT_INTG | ADC_CLK_AUTO
        #define PARAM2  ADC_VREF_AVDD_AVSS | ADC_OFFSET_CAL_DISABLE | ADC_SCAN_OFF | ADC_ALT_INPUT_OFF
        #define PARAM3  ADC_CONV_CLK_INTERNAL_RC | ADC_SAMPLE_TIME_15
        #define PARAM4 ENABLE_AN9_ANA
        #define PARAM5 SKIP_SCAN_ALL
        SetChanADC10( ADC_CH0_NEG_SAMPLEA_NVREF | ADC_CH0_POS_SAMPLEA_AN9 );
        OpenADC10( PARAM1, PARAM2, PARAM3, PARAM4, PARAM5 );
        EnableADC10();
        AcquireADC10();
        while ( ! mAD1GetIntFlag() ) {
                // wait for the first conversion to complete so there will be vaild data in ADC result registers
        }
        channel4 = ReadADC10(ADC1BUF9);
        return Decodifica_ADC_Micro(channel4);
        mAD1ClearIntFlag();
        CloseADC10();
}
```

Funzione

ReadThreshold

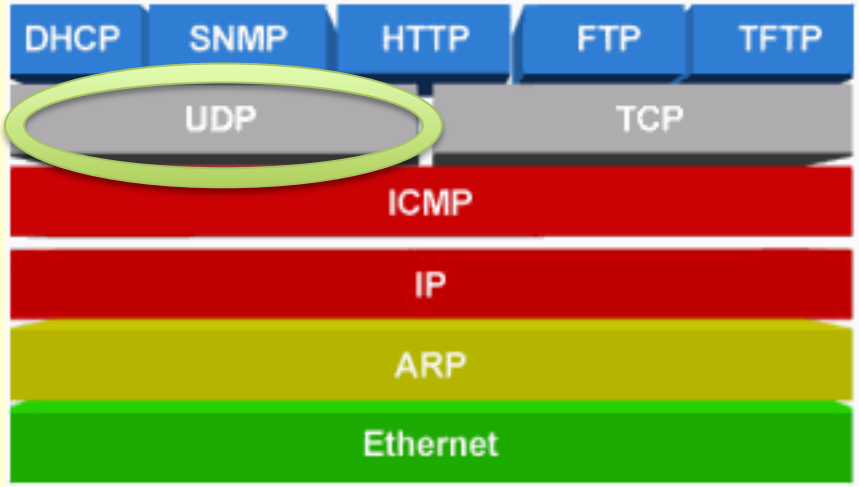
Sviluppi futuri, estensione sulla rete

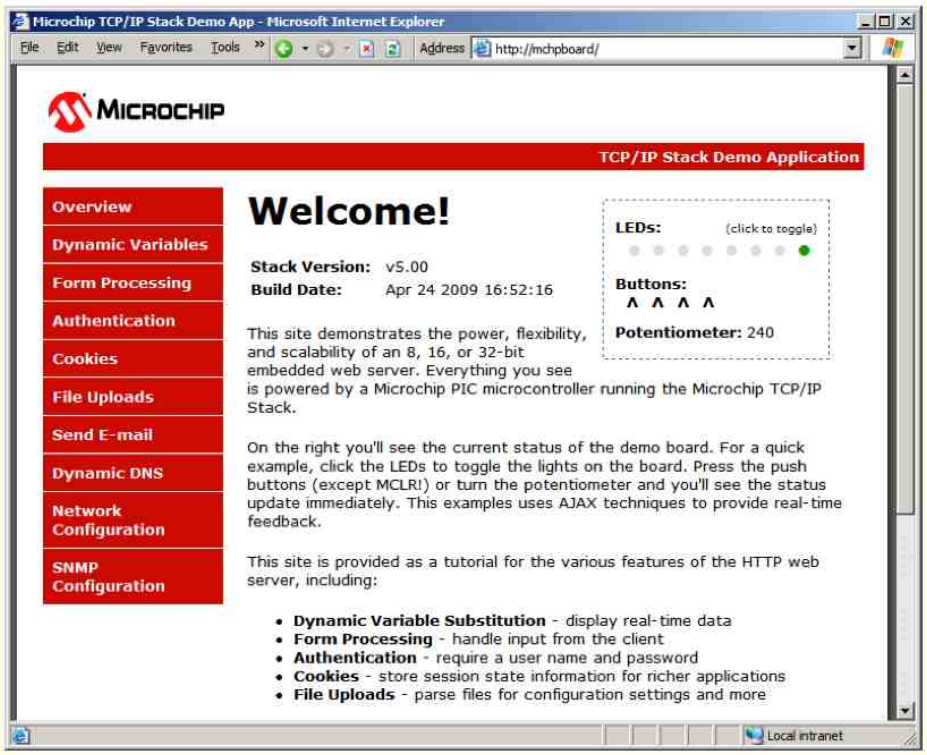
- Il microcontrollore gestirà un interfaccia di rete, tramite la quale potrà trasmettere e ricevere dati.
- L'integrazione con un'adatta architettura di rete permetterà il completo cablaggio di un intero deposito e la diffusione su di una rete locale di tutti i dati necessari.
- Un applicazione potrà gestire ogni singola centralina leggendo i valori misurati ed impostando i corretti settaggi.



Lo stack TCP/IP

- Stack fornito direttamente dalla casa produttrice del PIC.
- L'obiettivo è quello di escludere i servizi che non sono necessari e mantenere solamente la comunicazione UDP.






GRAZIE.