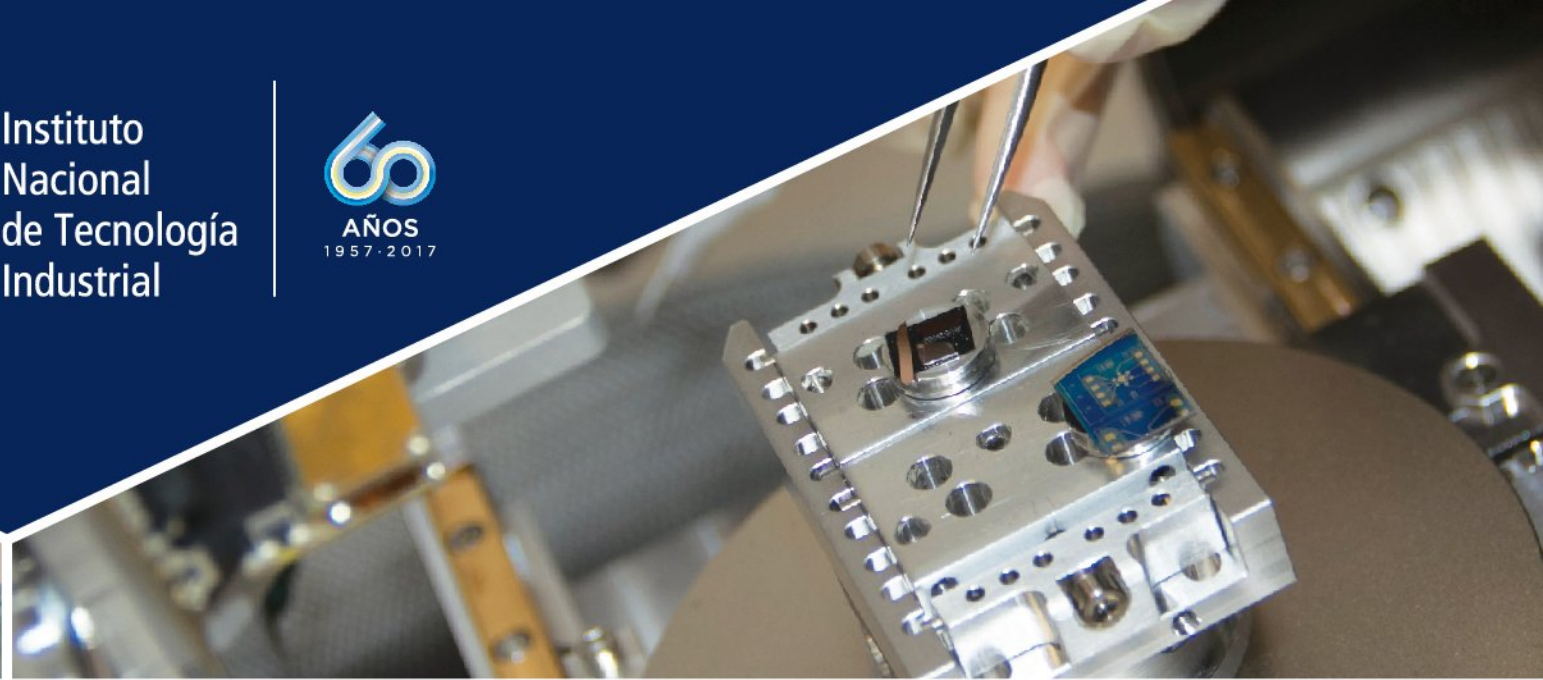




INTI

Instituto
Nacional
de Tecnología
Industrial



Instituto Nacional de Tecnología Industrial

Arduino por dentro: ¿Cómo hago un Arduino?

Disertante: Salvador E. Tropea
Centro de Micro y Nanoelectrónica (CMNB)



Ministerio de Producción
Presidencia de la Nación

SASE 2017



INTI

Instituto
Nacional
de Tecnología
Industrial



AÑOS
1957-2017



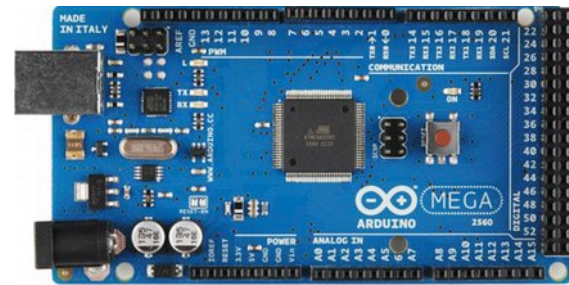
Ministerio de Producción
Presidencia de la Nación

Breve introducción y ventajas del Arduino



ARDUINO

- Es la plataforma embebida más popular de los últimos años.
 - Usada por técnicos y hobbistas, pero también artistas.
 - Muy bajo costo (\$ 142 Arduino UNO en ML*, \$75 el nano)
 - Para algunos es sinónimo de electrónica (Mercado Libre)
- Gran variedad de opciones “compatibles”

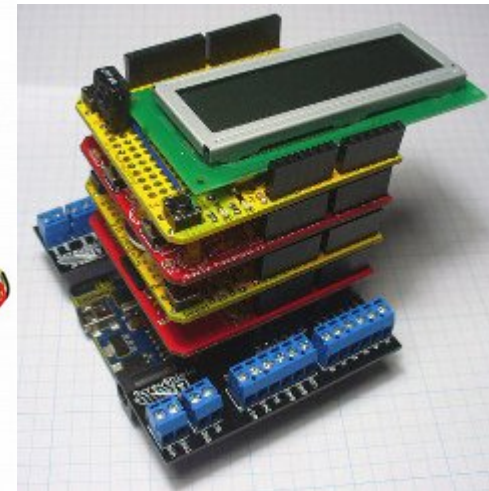
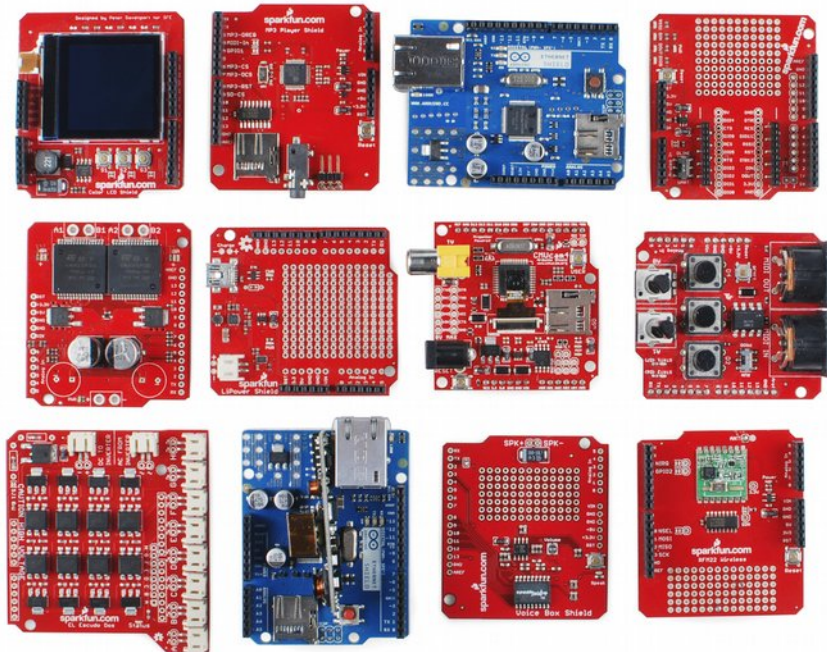


(* Clón, el original cuesta U\$S 25 FOB



ARDUINO Accesorios y Shields

- Gran variedad de accesorios
- Los *shields* permiten extender su funcionalidad
- Bajo precio





ARDUINO Programación

- Se usa lenguaje “Sketch” (C++ con ayuditas)
- Compila a código nativo: buena performance
- API simple de usar que abstrae de varios detalles de hard
- IDE de uso simple
- Muchas bibliotecas y ejemplos disponibles

```
Archivo Editar Programa Herramientas Ayuda
Adafruit_touchpaint
interface (RST is optional)
Adafruit invests time and resources providing this open source code,
please support Adafruit and open-source hardware by purchasing
products from Adafruit!

Written by Limor Fried/Ladyada for Adafruit Industries.
MIT license, all text above must be included in any redistribution
*****/

#include <Adafruit_ILI9341_flat.h>
#include <Adafruit_STMPE610_b.h>

// This is calibration data for the raw touch data to the screen coordinates
#define TS_MINX 150
#define TS_MINY 100
```

Estudiarlo:
¿Por qué?
¿Para qué?

Razones para estudiar la arquitectura

- Para poder resolver problemas en el uso
- Para enseñar a usarlo con conocimiento profundo
- Por curiosidad
- Para hacer un Arduino
- ¿Hacerlo? Si sale tan barato ...
 - Tiene que tener alguna ventaja especial
 - Ya sea didáctica
 - Ya sea operativa

LA LATTUINO

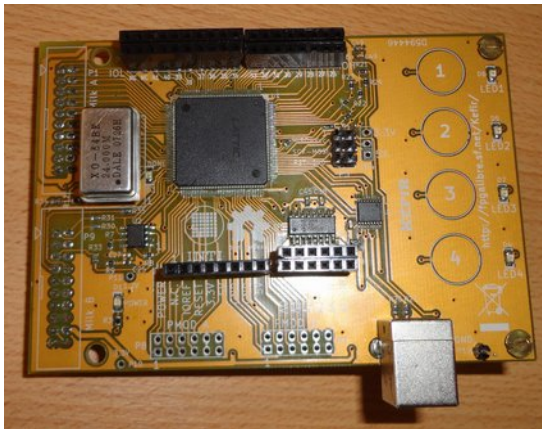
<http://fpgalibre.sf.net/Lattuino/>

KÉFIR

<http://fpgalibre.sf.net/Kefir/>

Caso de estudio

- Lattuino
- Compatible con Arduino UNO
- Basado en hardware reconfigurable (FPGA)
 - ¡Podemos crear nuestros periféricos a medida!
- Primera implementación de hardware abierto c/FPGA
- Le agrega valor a la placa Kéfir I (iCE40HX4K)



Caso de estudio

- Ahora que tenemos la excusa ... levantemos el capot
- Arduino “under the hood”



Caso de estudio

- Bueno ... para ser más realistas ...



Arduino

“Under the hood”

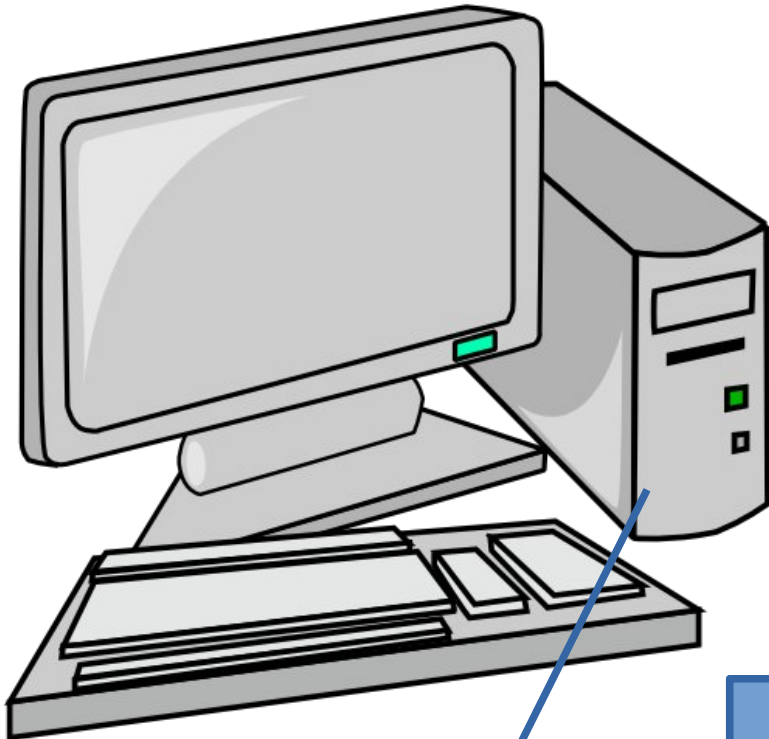
Aspectos a tener en cuenta

- Arduino/Genuino son marcas registradas por lo que en realidad estaremos haciendo algo “compatible con Arduino”
- Detalles en <https://www.arduino.cc/en/Main/Policy>
- No hay una norma que diga que es necesario
- De hecho los distintos Arduinos tienen grandes diferencias
- Probablemente nos convenga centrarnos en uno particular
 - El Arduino UNO r3 es muy popular
 - Lattuino es compatible con UNO r3
- Por lo tanto: no es obligatorio cumplir con todos los detalles, pero cuanto más se parezca más se puede reusar de lo disponible.

Aspectos a tener en cuenta

- CPU (incluyendo los periféricos)
- Conectores (disposición, señales, niveles de tensión)
- Mecanismo de programación
- API (biblioteca de software base)
- Integración con la IDE de Arduino
- Otros detalles menores

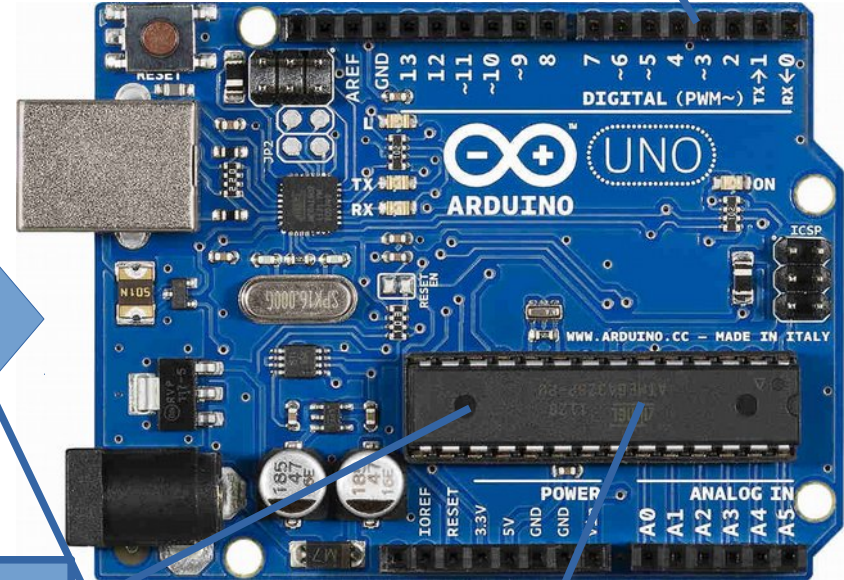
Aspectos a tener en cuenta



Integración c/IDE API



Conectores



Bootloader

CPU y periféricos

CPU y periféricos

CPU

- Arduino está basado en los AVR de generación 2 y posterior
- La línea más común es la ATmega
- El UNO r3 usa un ATmega328 (derivado del ATmega32)
- Es posible utilizar otra CPU, de hecho muchos productos comerciales lo hacen pero ...
 - Es necesario portar la API
- Cualquier diferencia en los periféricos puede derivar en una adaptación de la API.
- Hay muchas bibliotecas que solo funcionan con AVR, y algunas solo con algunos AVR en particular

Conectores

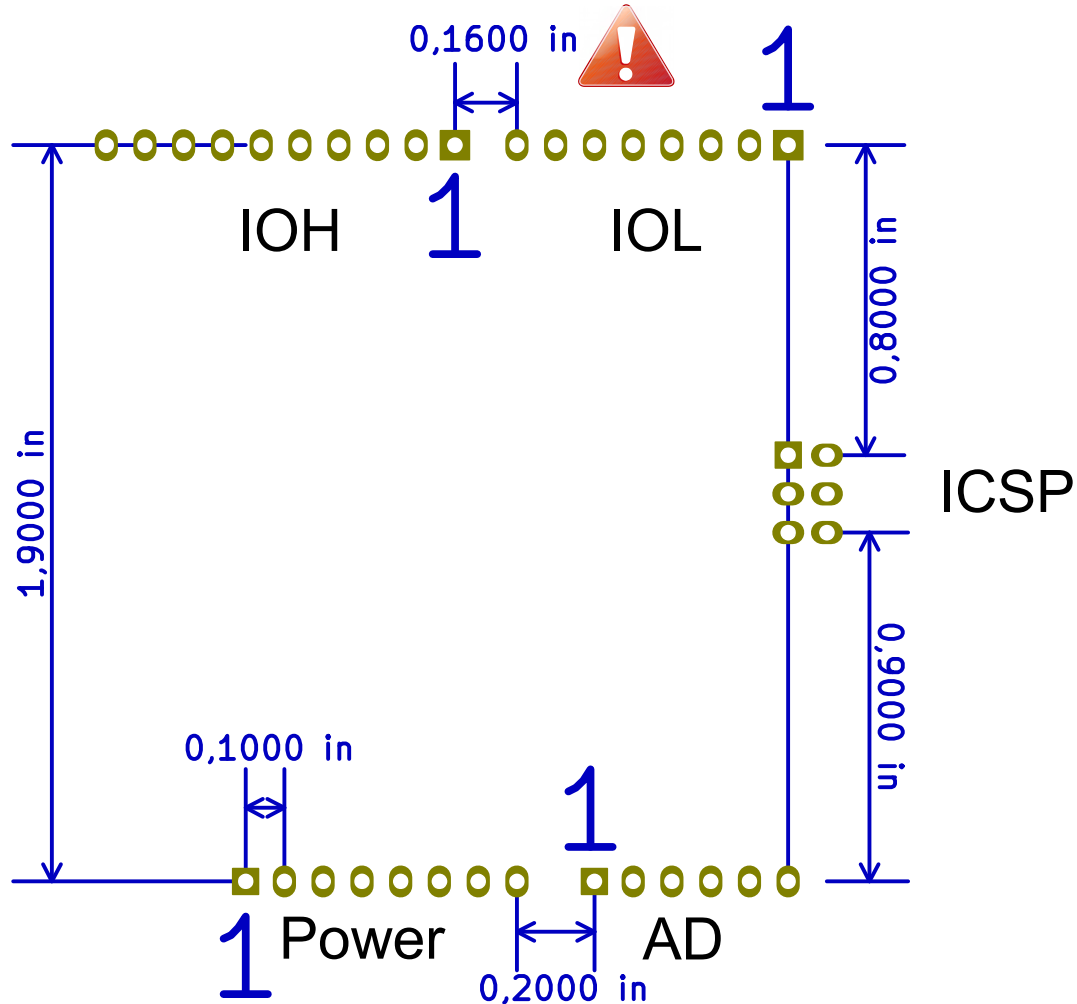
Conectores

- Si queremos poder usar los *shields* de Arduino es muy importante.
- Hay mucha información y explicaciones para los conectores estandarizados de Arduino.
- Existen 2 juegos de conectores
 - Tradicionales (p.e. Arduino UNO)
 - Extendidos (p.e. Arduino MEGA)

Conectores tradicionales de Arduino

- Es un conjunto de 5 conectores:
 - Power (8x1, originalmente 6x1)
 - IOL (8x1)
 - IOH (10x1, originalmente 8x1)
 - AD (6x1)
 - ICSP (3x2)
- Su disposición en el PCB es un tanto caprichosa

Conectores tradicionales de Arduino



Lattuino implementa esta disposición

Conectores tradicionales de Arduino

■ Power

- 1) N.C. *
- 2) IOREF (5 V, hmmm ...) *
- 3) Reset (activo bajo)
- 4) 3,3 V
- 5) 5 V
- 6) GND
- 7) GND
- 8) Vin (la que alimenta a la placa)

* No estaban en las versiones viejas

Conectores tradicionales de Arduino

■ IOL

- 1) IO0 (opcionalmente Rx)
- 2) IO1 (opcionalmente Tx)
- 3) IO2 (fuente de interrupción INT0)
- 4) IO3 (fuente de interrupción INT1) *
- 5) IO4
- 6) IO5 *
- 7) IO6 *
- 8) IO7

* Pueden usarse como PWM

Aclaraciones () aplican a Arduino UNO

Conectores tradicionales de Arduino

■ IOH

- 1) IO8
- 2) IO9 *
- 3) IO10 *
- 4) IO11 *
- 5) IO12
- 6) IO13
- 7) GND
- 8) AREF
- 9) SDA **
- 10) SDL **

* Pueden usarse como PWM

** No estaban en versiones viejas, en UNO r3 se comparten con AD4/5

Conectores tradicionales de Arduino

- AD
 - 1) AD0
 - 2) AD1
 - 3) AD2
 - 4) AD3
 - 5) AD4 (SDA)
 - 6) AD5 (SCL)

Aclaraciones () aplican a Arduino UNO

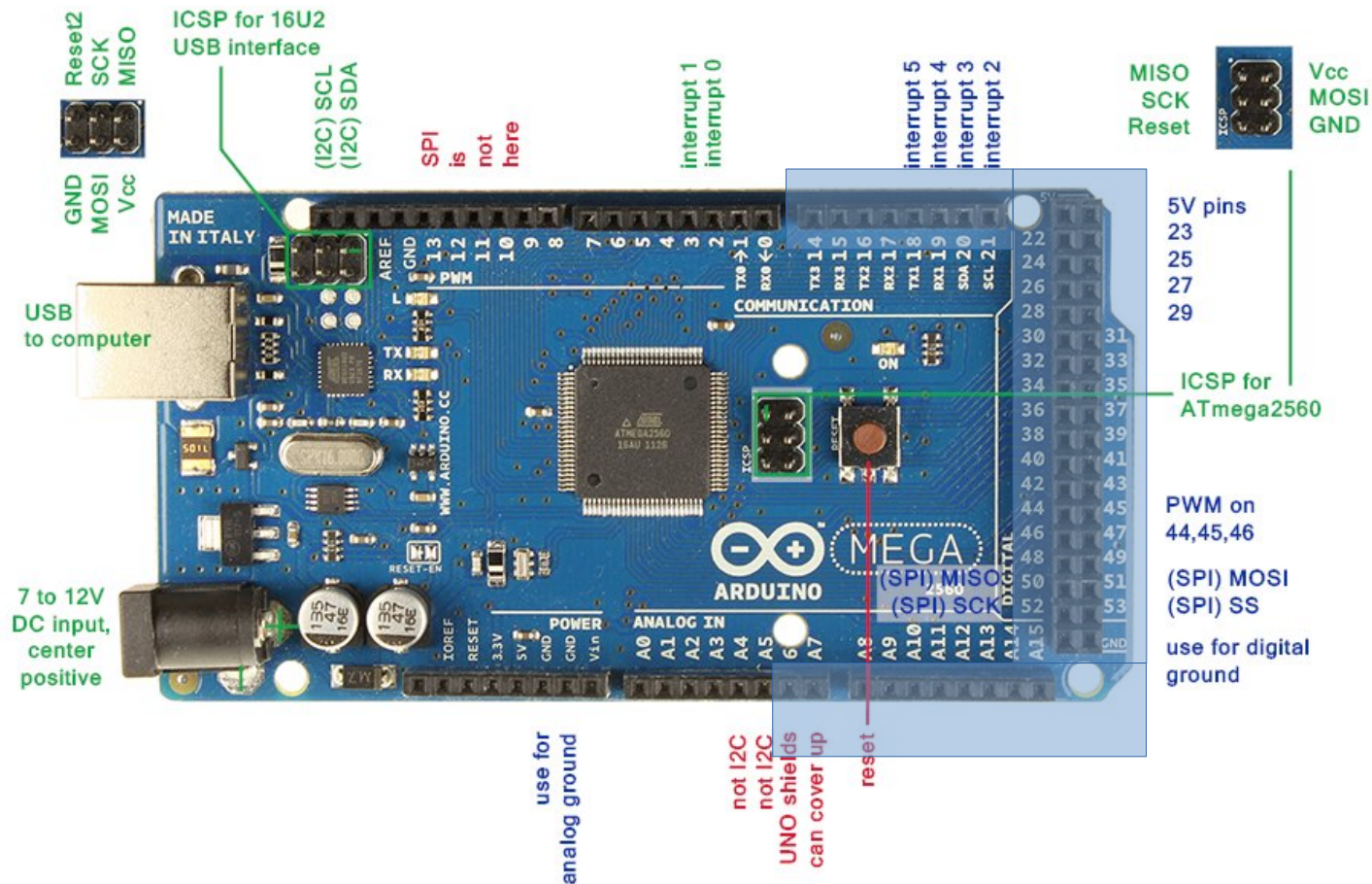
Conectores tradicionales de Arduino

- ICSP [SPI/debug flasheo bootloader]
 - 1) MISO (IO12)
 - 2) 5 V
 - 3) SCK (IO13)
 - 4) MOSI (IO11)
 - 5) Reset [activo bajo]
 - 6) GND

Aclaraciones () aplican a Arduino UNO

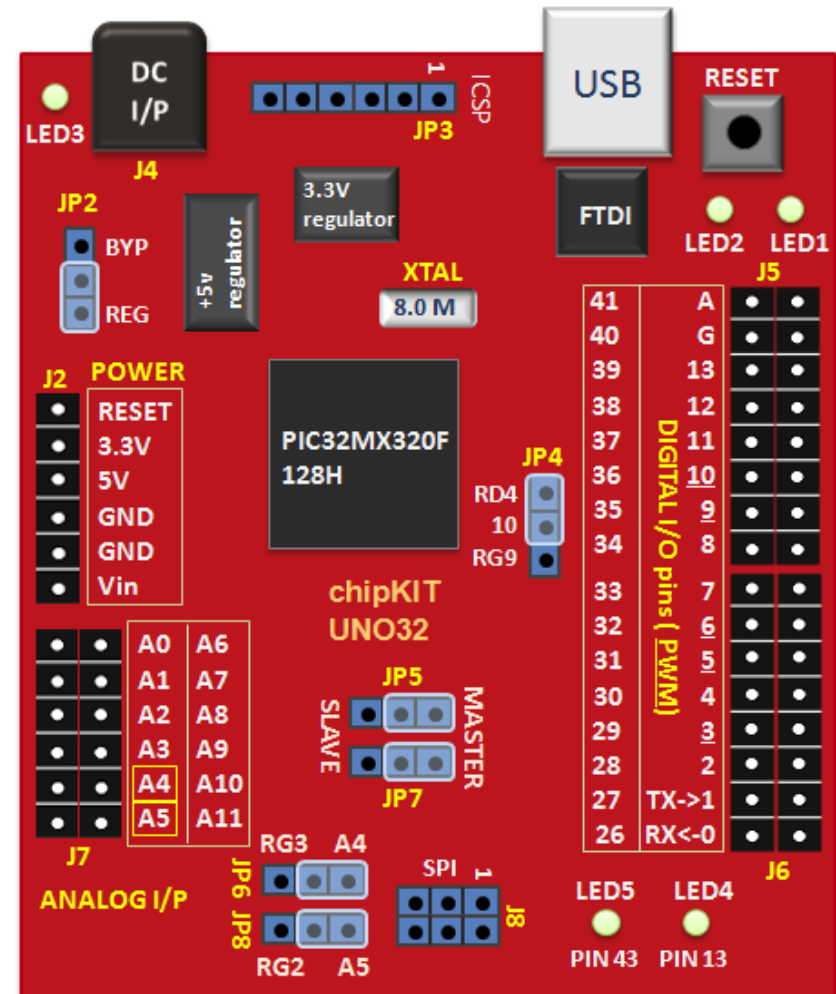
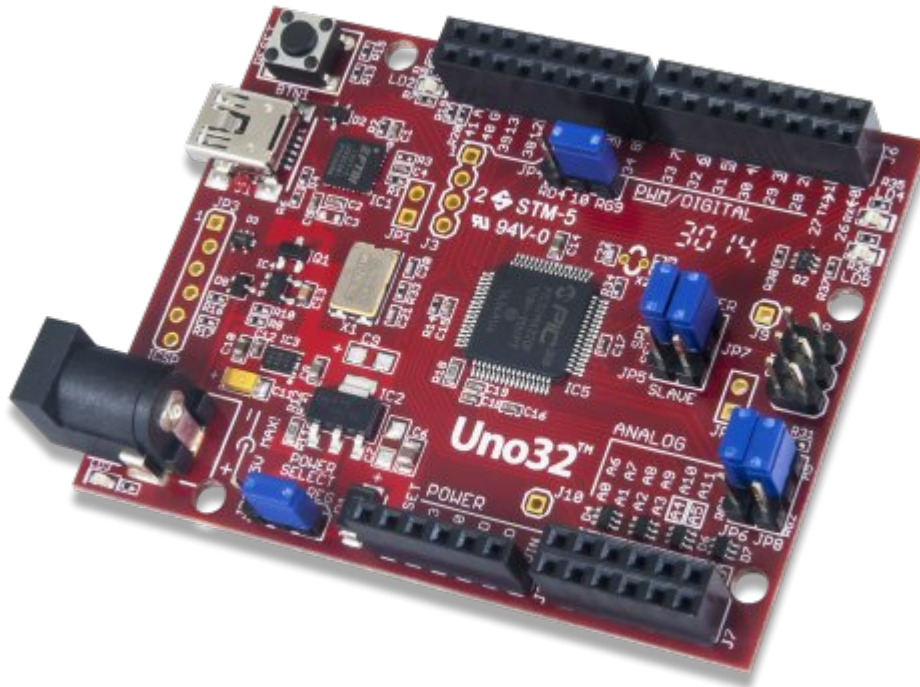
Conectores extendidos de Arduino

- El Arduino MEGA agrega:



Conectores extendidos de Arduino


- Digilent usa:



Kéfir I implementa esta disposición



Conectores tradicionales de Arduino

- Los Arduinos originales eran de 5 V
- Actualmente hay muchos de 3,3 V
- Normalmente toleran 5 V
- Lattuino usa 3,3 V y NO tolera 5 V 
- Se asume que las salidas pueden manejar corrientes de hasta algunas decenas de mA

Comunicación con la PC (*bootloader*)

Mecanismo de comunicación

- Normalmente los Arduinos se configuran y comunican con la PC utilizando RS-232 sobre USB
- Los Arduinos poseen un firmware denominado *bootloader*
- El procedimiento normal de configuración que usa la IDE es comunicándose con dicho *bootloader*
- La idea es que el *bootloader* se aloja “al fondo” de la memoria, y la CPU se configura para ejecutarlo luego de un RESET
- Esto resta un poco de memoria de programa disponible (328 *words* en el caso de Lattuino)

Mecanismo de comunicación

- El protocolo implementado es el STK500, usado por Atmel para sus kits de desarrollo.
- El *bootloader* espera unos segundos, si no recibe comandos pasa el control al programa ya cargado en la *flash*
- La PC envía comandos de 1 byte, seguidos por argumentos y terminados en un espacio (0x20), a los cuales la CPU responde.
- Las respuestas típicas consisten en 1 byte que indica si la CPU está en sincronismo (0x14), una cantidad de bytes variables dependiendo del comando y 1 byte indicando el estado (0x10==OK)

Mecanismo de comunicación

- Si se perdieran bytes ambos quedarían desincronizados. Para volver a sincronizarse la PC envía el comando GET_SYNC (“0”) hasta obtener una respuesta que diga IN_SYNC (0x10) y OK (0x14)
- La nota de aplicación AVR068 describe el protocolo completo (<http://www.atmel.com/images/doc2591.pdf>)
- No es necesario que el *bootloader* implemente todos los comandos, a continuación se listan los que implementa el bootloader de los Arduinos basados en ATmega8
- Lattuino implementa estos comandos

Mecanismo de comunicación

Comando	Hexa	Argumentos	Respuesta
GET_SYNC	30	Ninguno	Nada
GET_SIGN_ON	31	Ninguno	“AVR ISP”
SET_PARAMETER	40	Nro parámetro, Valor a setear	Nada
GET_PARAMETER	41	Nro parámetro	Valor del parámetro
SET_DEVICE	42	20 valores	Nada
SET_DEVICE_EXT	45	5 valores	Nada
ENTER_PROGMODE	50	Ninguno	Nada
LEAVE_PROGMODE	51	Ninguno	Nada

Mecanismo de comunicación

Comando	Hexa	Argumentos	Respuesta
CHIP_ERASE	52	Ninguno	Nada
LOAD_ADDRESS	55	Byte bajo, byte alto	Nada
UNIVERSAL	56	4 bytes	1 byte
PROG_PAGE	64	Byte alto largo, byte bajo largo, E/F, N bytes	Nada
READ_PAGE	74	Byte alto largo, byte bajo largo, E/F	N bytes
READ_SIGN	75	Ninguno	3 bytes
READ_OSCCAL	76	Ninguno	1 byte

Mecanismo de comunicación

- SET_PARAMETER puede ser ignorado
- Los únicos parámetros importantes a responder en GET_PARAMETER son:
 - HW_VER (0x80): 2
 - SW_MAJOR (0x81): 1
 - SW_MINOR (0x82): 18
 - Al resto se le puede devolver 0
- SET_DEVICE y SET_DEVICE_EXT setean varios parámetros de programación de la CPU. Están pensados para *firmwares* genéricos. Si el *firmware* es *custom* pueden ignorarse.

Mecanismo de comunicación

- ENTER_PROGMODE inicia la programación del dispositivo. Esto suspende el timer que pasa el control al programa antes cargado.
- LEAVE_PROGMODE la finaliza.
- CHIP_ERASE se usa para borrar toda la memoria de programa (menos el bootloader 😊)
- LOAD_ADDRESS indica la dirección de memoria a partir de donde se va a descargar el programa (**words**)
- PROG_PAGE escribe una página completa de la flash. En el caso del ATmega8 jamás son más de 256 bytes.
 - El largo se expresa en bytes

Mecanismo de comunicación

- E=EEPROM F=Flash
- El orden de los datos enviados es LOW/HIGH
- READ_PAGE lee el contenido de una página.
- READ_SIG lee los 3 bytes que identifican la CPU
- READ_OSCCAL lee el valor de calibración del oscilador interno. Puede contestarse 0.
- UNIVERSAL está pensado para “cables” de programación, no *bootloaders*, y envía 32 bits por el SPI que conecta al cable con la CPU, la respuesta son los 8 bits respondidos por la CPU. Se puede responder siempre 0.

API

*(Application Programming
Interface)*

API base

- Siempre es necesario configurarla creando un archivo **pins_arduino.h**
- Si nuestra CPU y/o periféricos no son los de un AVR ya soportado es necesario portarla/ajustarla
- El Lattuino usa una CPU compatible con el AVR (generación “2.5”), pero no todos sus periféricos son iguales a los de un AVR
- La API está escrita en C++, C y assembler (C++ **básico**)
- La misma se compila cada vez que compilamos un programa
- Se tomó como base la que viene con la IDE 1.8.1

API base: `pins_arduino.h`

Algunas definiciones importantes:

- `NUM_DIGITAL_PINS` número de pines digitales, incluyendo cualquier LED interno
- `NUM_ANALOG_INPUTS` número de entradas analógicas
- `LED_BUILT_IN` si tenemos un LED en la placa que queramos que esté disponible para el usuario definimos esta constante con el número de pin donde está conectado
- Para las entradas analógicas es necesario definir constantes `const uint8_t Ax` con el valor del canal físico para esa entrada
- Las constantes `SS`, `MOSI`, `MISO` y `SCK` definen que pines implementan el puerto SPI (en realidad parte de una biblioteca)

API base: pins_arduino.h

Puertos de entrada/salida digital en AVR:

- Es necesario definir unas tablas que le permiten a la API traducir un número pasado por el usuario al puerto físico donde se encuentra.
- `digital_pin_to_port_PGM` mapea el número a el número de puerto donde se encuentra (PA, PB, PC, etc.)
- `digital_pin_to_bit_mask_PGM` mapea el número a la máscara usada para filtrar el bit en su puerto (`_BV(bit)`)
- Como los puertos PA, PB, etc. no están en los mismos registros de cada AVR hay 3 tablas extra.

API base: pins_arduino.h

Puertos de entrada/salida digital en AVR:

- `port_to_mode_PGM` traduce el número de puerto al puntero al registro `DDRx` asociado (`PB => DDRB`)
- `port_to_output_PGM` hace lo mismo con el registro `PORTx`
- `port_to_input_PGM` es para el registro `PINx`
- Para definir que pines pueden funcionar como PWM se usa una tabla `digital_pin_to_timer_PWM_PGM` que mapea el número de puerto al número de timer asociado (`TIMER0A`, `TIMER2B`, etc.)

API base

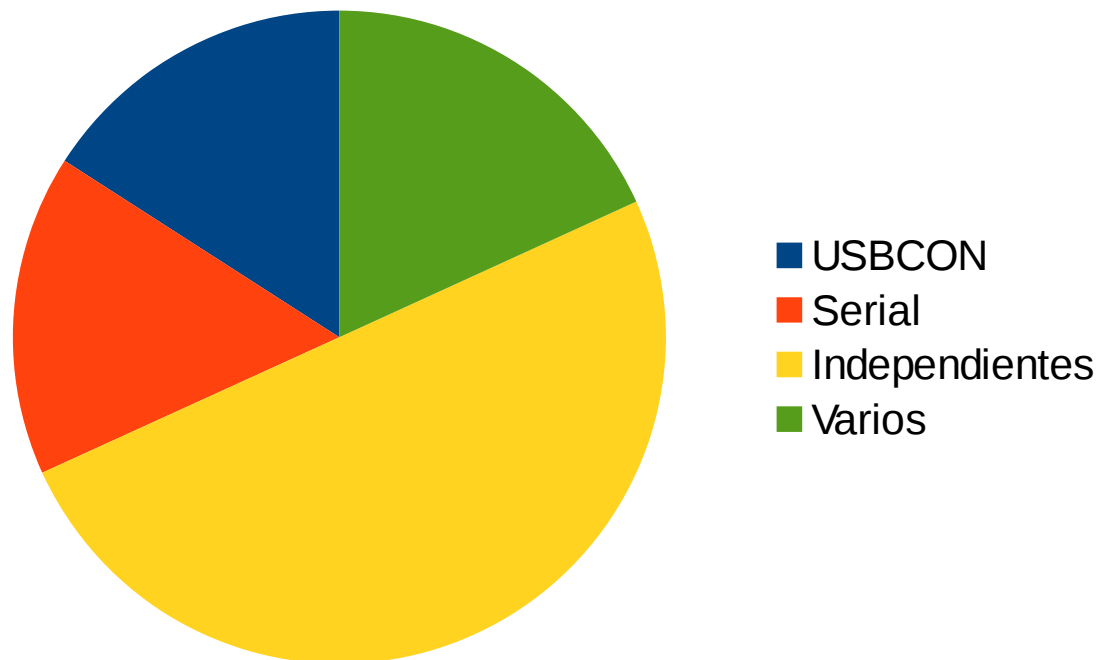
- La API de Arduino se basa en el “Wiring Framework”:
<http://wiring.org.co/>
- Son 44 archivos
- CDC.cpp, PluggableUSB.cpp, PluggableUSB.h, USBAPI.h, USBCore.cpp, USBCore.h y USBDesc.h (7) son relevantes solo para implementaciones que implementan comunicación sobre USB directa (USBCON).
- HardwareSerial*.* (7) implementan hasta 4 puertos serie
- Tone.cpp implementa el generador de tonos (usualmente basado en el timer 2)

API base

- WInterrupts.c implementa las interrupciones externas (p.e. INT0/INT1)
- wiring_analog.c implementa las entradas y salidas analógicas (ADC y PWM)
- wiring_digital.c implementa las digitales
- wiring_pulse.* (2) implementa la medición del ancho de pulsos (código en assembler para pulsos cortos)
- wiring.c implementa las demoras de tiempo y la rutina de inicialización de los periféricos
- wiring_private.h contiene definiciones relacionadas con varios de los fuentes antes mencionados

API base

- Los 22 fuentes restantes no dependen en forma directa del hardware.



API extendida

- Adicionalmente se incluyen 5 bibliotecas “base”
 - EEPROM: permite leer/escribir la EEPROM
 - HID: permite implementar dispositivos HID sobre USB
 - SPI: comunicación SPI
 - SoftwareSerial: RS-232 por software
 - Wire: comunicación I2C
- En el caso del Lattuino solo se soportan SPI y SoftwareSerial, sin cambios en el código original

Integración con la IDE de Arduino

Integración con la IDE de Arduino

- El número de placas con soporte para la IDE es enorme:
<https://github.com/arduino/Arduino/wiki/Unofficial-list-of-3rd-party-boards-support-urls>
- Por lo que son plug-ins que se agregan a la IDE
- El plug-in puede ser algo tan simple como la descripción de que AVR posee la placa, y compatible con cuál otra es, a ser algo muy complejo que incluya un tool-chain para la CPU usada.
- Como la IDE de Arduino está escrita en Java la forma de describir los plug-ins es usando ... JSON (Java Script Object Notation)

Integración con la IDE de Arduino, .JSON

```
{  
  "packages": [  
    {  
      "name": "Lattuino",  
      "maintainer": "FPGA Libre",  
      "websiteURL": "http://fpgalibre.sf.net/",  
      "email": "set@ieee.org",  
      "help": {  
        "online": ""  
      },  
    },  
  ],  
}
```

Inicio del paquete

Con este nombre aparecen
agrupadas en el menú
Herramientas|Placa

Integración con la IDE de Arduino, .JSON

```
"platforms": [  
  {  
    "name": "Lattuino 1",  
    "architecture": "avr",  
    "version": "1.0.4",  
    "category": "Arduino",  
    "url": "http://fpgalibre.sf.net/Lattuino/lattuino_1-1.0.4.tar.bz2",  
    "archiveFileName": "lattuino_1-1.0.4.tar.bz2",  
    "checksum": "SHA-256:931478e01be9be1cd568ad17266cb325527f137c6a4b3829c5dbf31a995f247b",  
    "size": "56055",  
    "boards": [  
      {"name": "Kefir I"},  
      {"name": "iCE Stick"}  
    ]  
  }  
]
```

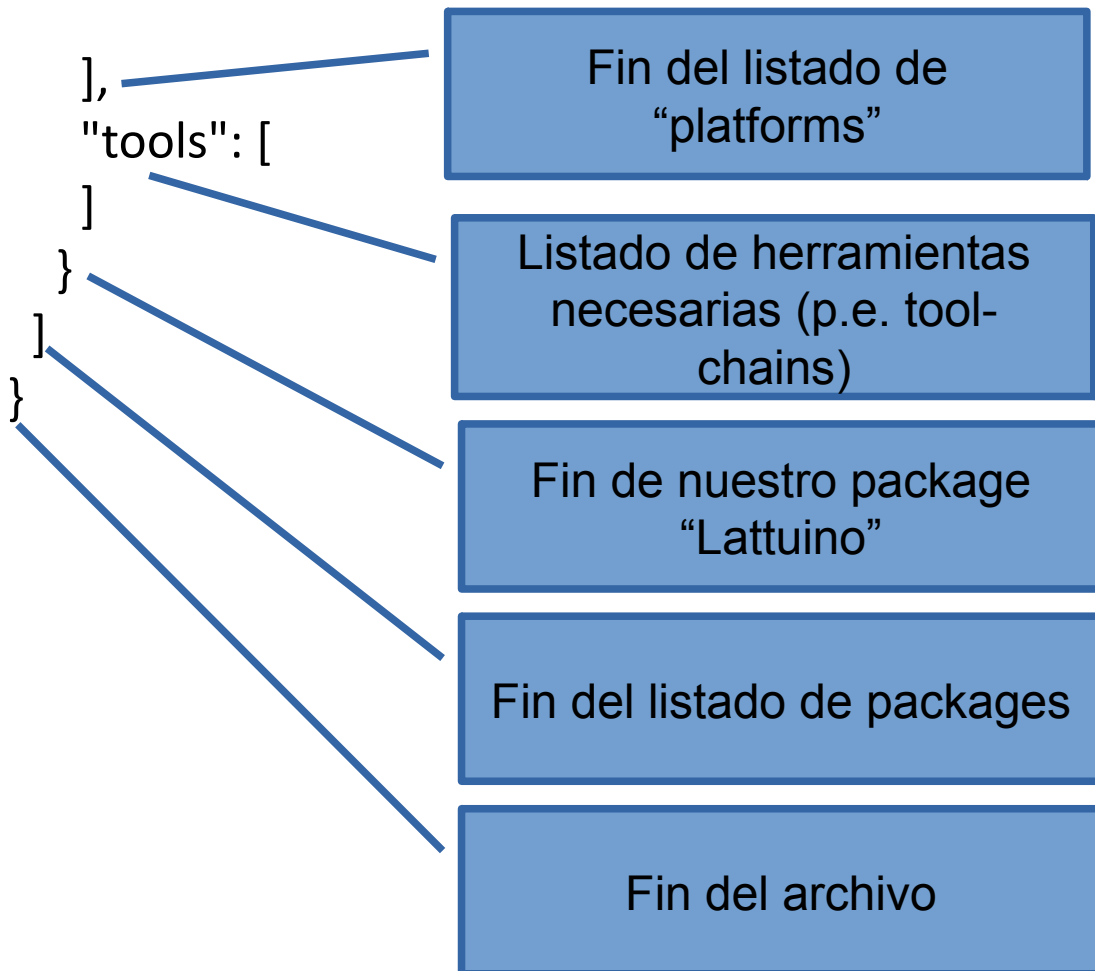
Una versión del plug-in, otras separar con “ , ”

De donde descarga el tarball con el plug-in

Salida de sha256sum computada sobre archiveFileName

Listado de placas soportadas por este plug-in

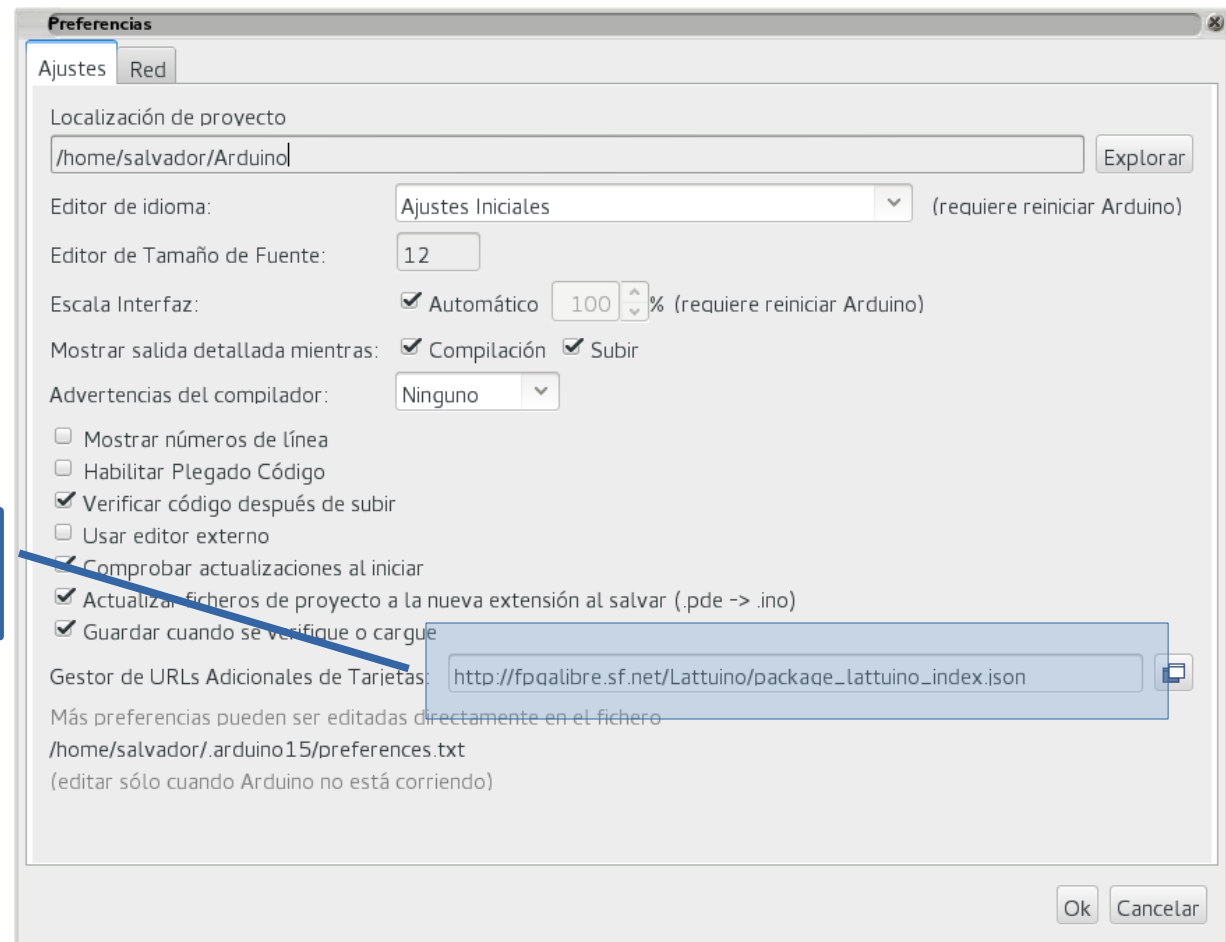
Integración con la IDE de Arduino, .JSON



Integración con la IDE de Arduino

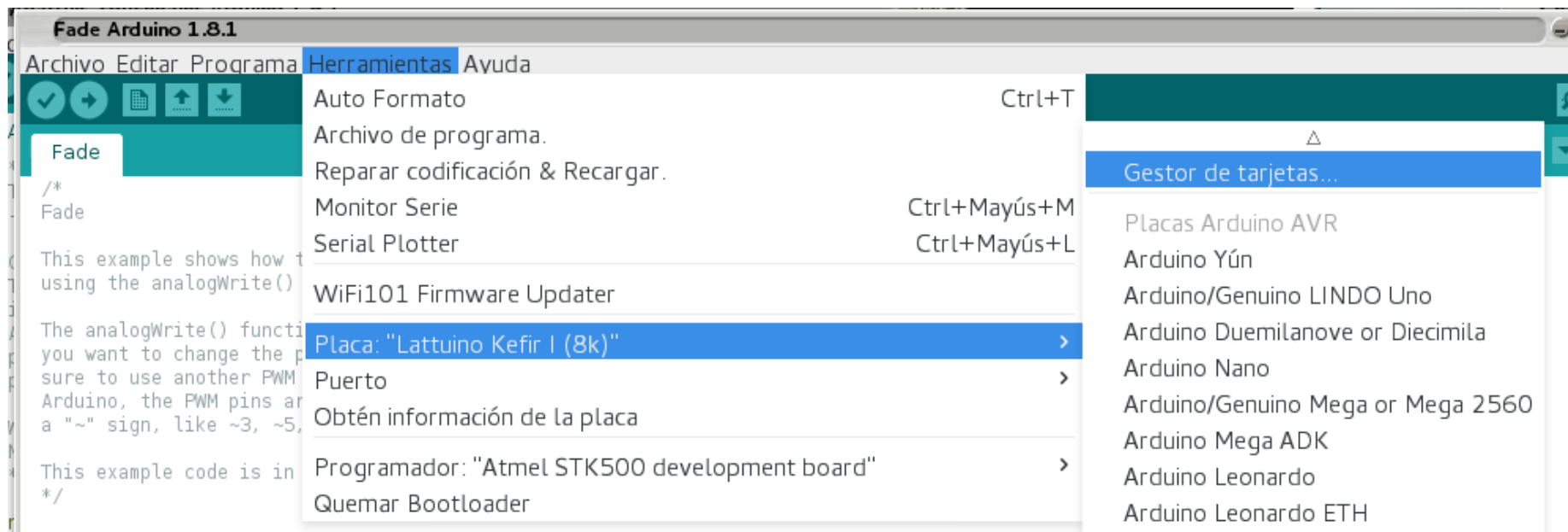
- El .JSON lo subimos a internet y configuramos la IDE para que lo lea.
- En Archivo | Preferencias

URL del .JSON

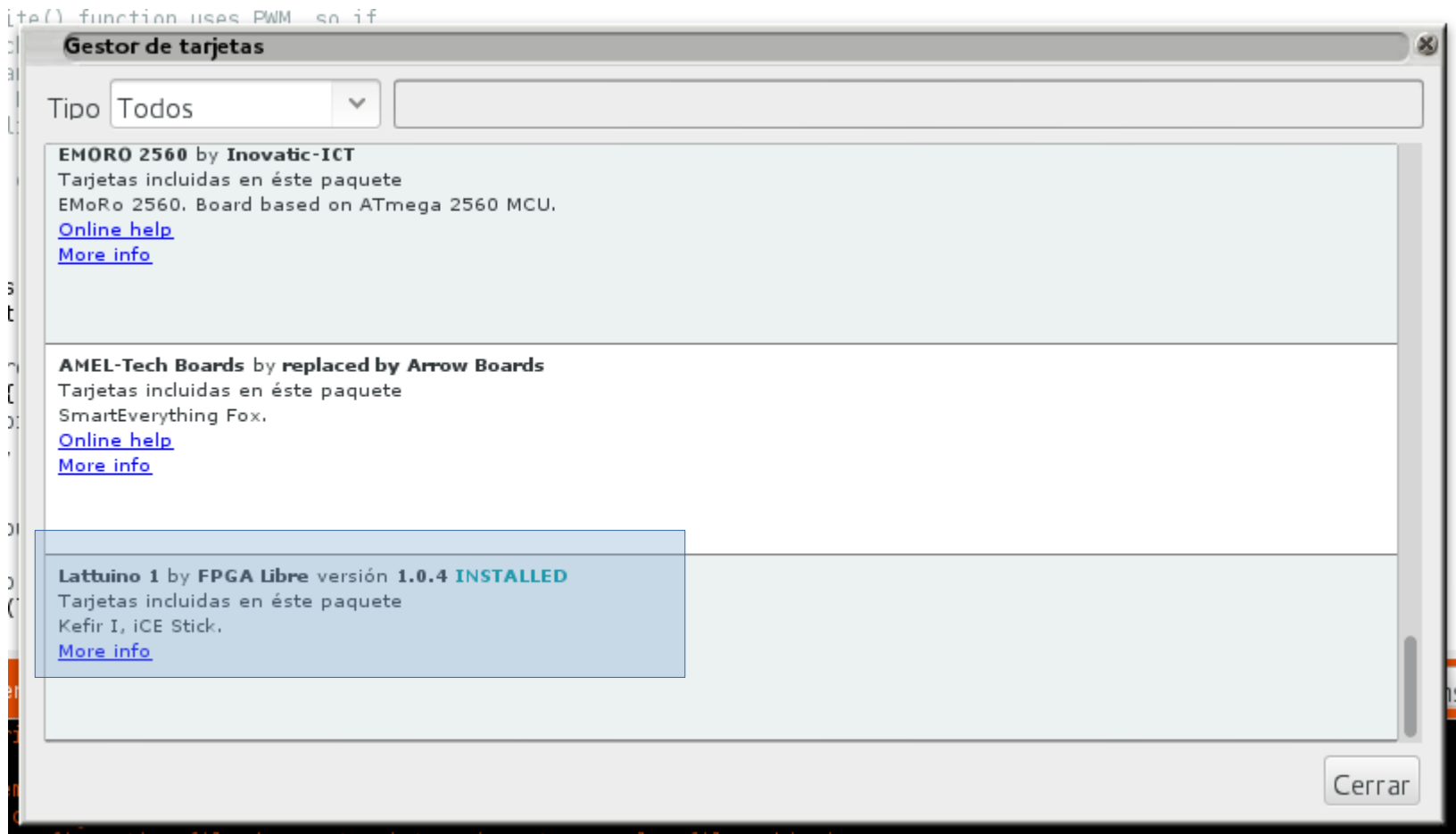


Integración con la IDE de Arduino

- Con esto el gestor de tarjetas podrá descargar el JSON
- Lo vemos en:



Integración con la IDE de Arduino



Integración con la IDE de Arduino

- ¿Qué va dentro del plug-in?
- Todo el contenido del plug-in va dentro de un directorio con el mismo nombre que la versión del mismo (p.e. 1.0.4)
- Dentro de ese directorio van las distintas cosas que le agregamos a la IDE.
- Estas cosas se “agregan” al directorio “hardware”, en el subdirectorio que declaramos como “architecture” en el JSON
- En nuestro caso hardware/avr/
- “Agregan” porque en realidad complementan, no se copian
- A continuación se describen varias cosas importantes a incluir.

Integración con la IDE de Arduino: boards.txt

- Las “placas” soportadas por nuestro plug-in se describen en un archivo *boards.txt* que funcionará como extensión del que trae Arduino.
- Las “placas” que declaramos acá no tienen porque coincidir con los nombres usados en el JSON
- Todas ellas aparecerán en una sección del menú con el nombre de nuestro package (Lattuino en el ejemplo)

Integración con la IDE de Arduino: boards.txt

- Ejemplo de declaración:

```
latt1_8k.name=Lattuino Kefir I (8k)
latt1_8k.vid.0=0x0403
latt1_8k.pid.0=0x6010
latt1_8k.upload.tool=avrdude
latt1_8k.upload.protocol=arduino
latt1_8k.upload.maximum_size=7536
latt1_8k.upload.maximum_data_size=512
latt1_8k.upload.speed=115200
latt1_8k.build.mcu=attiny85
latt1_8k.build.f_cpu=24000000L
latt1_8k.build.board=AVR_UNO
latt1_8k.build.core=lattuino
latt1_8k.build.variant=lattuino
```

Integración con la IDE de Arduino: boards.txt

- Todas las líneas comienzan con un prefijo que identifica a esa placa (`latt1_8k`)
- `latt1_8k.name` es el nombre tal como aparece en el menú
- `vid` y `pid` son los Vendor y Product ID del dispositivo USB que reporta nuestra placa al conectarla.
- La sección `latt1_8k.upload` describe como subir un programa a la placa.
 - El protocolo arduino de `avrduide` es el más común y se refiere al STK500
 - `maximum_size` y `maximum_data_size` definen el tamaño de flash y RAM disponibles

Integración con la IDE de Arduino: boards.txt

- La sección `latt1_8k.build` describe como compilar
 - `mcu` es el nombre de la CPU (para gcc)
 - `f_cpu` es la frecuencia de la CPU, se lo pasa como `F_CPU` al código
 - `core` es el nombre de la arquitectura de nuestra placa. Lo normal es `arduino`, pero en nuestro caso como la API fue retocada le tenemos que dar un nombre diferente
 - `variant` es la variante dentro de la arquitectura, la más básica de `arduino` es `standard`. En nuestro caso es una variante de `lattuino` que se llama `lattuino`

Integración con la IDE de Arduino: platform.txt

- Este archivo define las reglas usadas para ejecutar las herramientas externas
- Esto incluye el compilado, enlazado y transferencia a la placa.
- Si nuestro core es arduino no necesitamos incluir este archivo en nuestro plug-in
- En el caso de Lattuino el core es diferente, pero en realidad la CPU es compatible con AVR (podemos usar avr-gcc) y la comunicación también (avrdude sirve)
- Por lo que el plug-in de Lattuino incluye una copia del archivo que viene con la IDE (salvo por name=Lattuino)

Integración con la IDE de Arduino: cores

- Si nuestro core es arduino no necesitamos incluir este directorio en nuestro plug-in
- Si necesitamos incluir una API modificada deberemos incluirlo.
- Dentro de este directorio debemos incluir un subdirectorio con el nombre que usamos para core, en nuestro caso `lattuino`
- Allí copiaremos toda la API modificada para nuestra placa
- En nuestro ejemplo es `cores/lattuino/`

Integración con la IDE de Arduino: variants

- Si nuestra `variant` es alguna de las incluidas con la IDE no necesitamos incluir este directorio en nuestro plug-in
- Dentro de este directorio debemos incluir un subdirectorio con el nombre que usamos para `variant`, en nuestro caso `lattuino`
- Allí copiaremos el archivo `pins_arduino.h`
- En nuestro ejemplo es `cores/lattuino/pins_arduino.h`

Integración con la IDE de Arduino: libraries

- Todas las bibliotecas extra que necesitemos para nuestras placas van en subdirectorios dentro de `libraries`
- En nuestro caso al haber modificado la API (un core diferente a `arduino`) es necesario incluir las implementaciones de las bibliotecas estándares que soportemos.
- Por lo que es necesario incluir copias de `SPI` y `SoftwareSerial`, aún cuando no hayan cambiado en nada.

Integración con la IDE de Arduino: estructura completa

1.0.4

```
|— boards.txt
|— cores
|   └─ lattuino
|       └─ abi.cpp
|           └─ ...
|               └─ WString.h
|— libraries
|   └─ SoftwareSerial
|       └─ keywords.txt
|           └─ ...
|   └─ SPI
|       └─ keywords.txt
|           └─ ...
|— platform.txt
└─ variants
    └─ lattuino
        └─ pins_arduino.h
```




INTI

Instituto
Nacional
de Tecnología
Industrial



AÑOS
1957-2017



Ministerio de Producción
Presidencia de la Nación

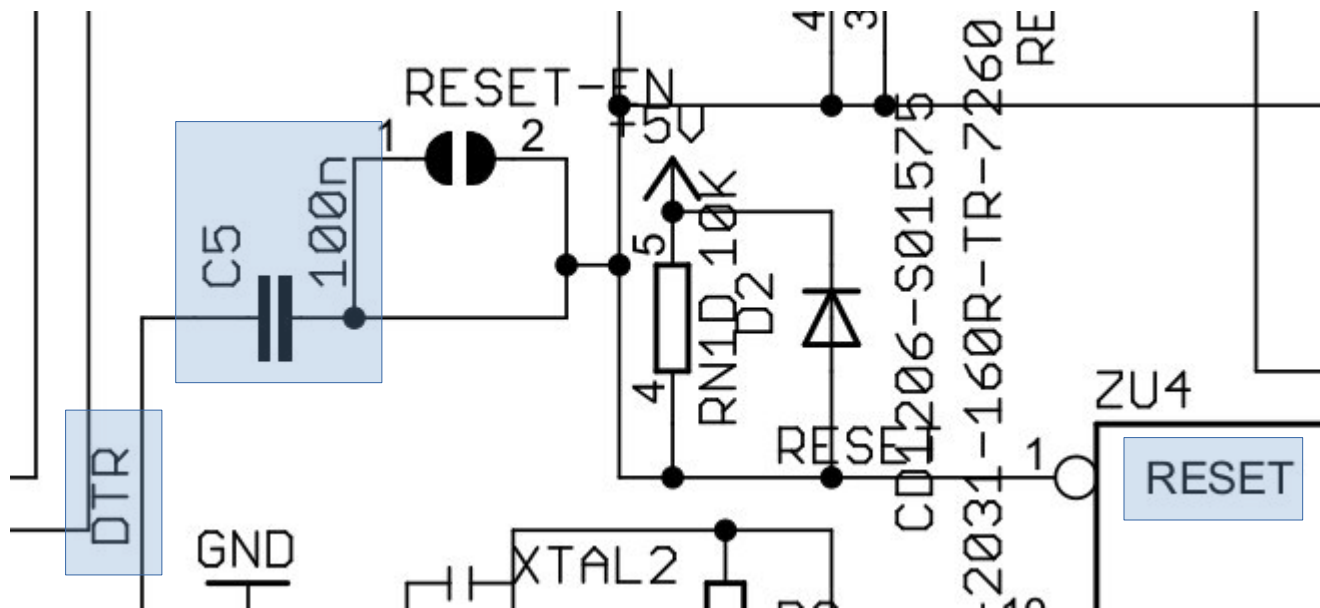
Otros detalles menores

Otros detalles menores: Arduino AutoReset

- La IDE soporta lo que se llama Arduino AutoReset
- Esto es algo que implementan los Arduinos modernos
- Cuando la IDE quiere transferir un programa genera un flanco de bajada en la línea DTR del RS-232, esto genera un RESET de la placa
- Luego del RESET la CPU carga el bootloader
- Por lo que la IDE puede transferir el programa casi inmediatamente luego de forzar un AutoReset
- En el Arduino UNO r3 esto se logra con un capacitor que conecta DTR a la línea de RESET

Otros detalles menores: Arduino AutoReset

- En la siguiente figura se muestra el circuito en cuestión



Otros detalles menores: Arduino Perogrullo ...

- El sitio de Arduino dice que debe incluir:
“all the electronic parts necessary to power and communicate with the microcontroller: regulator, clock crystal, USB-to-serial interface, and SPI programming interface for replacing the bootloader. Alternate design should include these elements as well, or show how to add them easily.”



INTI



¡Muchas Gracias!

Av. Gral. Paz 5445
(1650) San Martín
Buenos Aires, Argentina
+54-11-4724-6000 int. 6919
salvador@inti.gov.ar



Ministerio de Producción
Presidencia de la Nación

