Universidad Zaragoza
1542

Manager de Energía

ANEXOS

Jorge Ramos Guillen

2020

Contenido

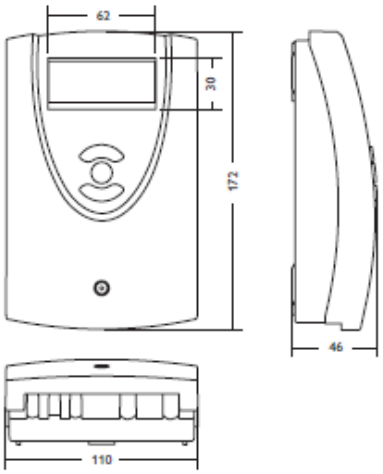
Anexo 1. Hojas de Características.

Presentamos la hoja de características principales de los componentes del proyecto, en orden: Pigeon, Resol, Contazara, Inversor Kostal, sonda DS1820.

CPU & memory		
SoC	BCM2835, ARM1176JZ-F core, 700MHz	
RAM memory	512Mbyte	
Flash memory	4Gbyte eMMC	
Power supply		
Supply voltage	15 ... 28V DC	
Power consumption	Conditions	Supply current @ 24V
	CPU 100% load, Ethernet 100Mbit active	96 mA
	CPU 100% load, Ethernet no active	86mA
	CPU 1% load, Ethernet no active	73 mA
	CPU 1% load, +3V3 peripherals switched off	36 mA
Interfaces		
Ethernet	1 x Ethernet 10/100-Mbit, Auto MDI-MDIX, RJ-45	
CAN	1 x CAN, MCP2515, terminal blocks	
1-WIRE	1 x 1-WIRE, DS2482S-100+, terminal blocks	
RS-232	1 x RS-232 (RXD, TXD, RTS, CTS), DB9 male	
RS-485	1 x RS-485, terminal blocks	
USB	3 x USB host 2.0 Type-A, 1 x Mini USB 2.0 Type B	
Inputs & Outputs		
Digital opto-isolated inputs	Channels	8
	Low-level input voltage	0 ... +5 V DC
	High-level input voltage	+10 ... +28V DC
	Isolation voltage	5 kV_{RMS}
	Input resistance	>=10kΩ
Dry contact inputs	Channels	4
Open drain outputs	Channels	8
	Maximum current	500 mA
	Maximum voltage	28 V DC
Analog inputs	Channels	4
	Voltage Range	0 ... +10V
	Resolution	10-bit
Analog outputs	Channels	2
	Voltage Range	0 ... +10V
	Resolution	10-bit
5V output DC	Total maximum current	1 A
	Note: Total maximum current is the current of +5V DC connector output and all USB +5V outputs	
Terminal blocks	Wire range	0.5 - 1.5 mm^2, 28 -16 AWG
	Torque	0.2 Nm
	Strip length	7 mm
Standards		
EU standard	EN 61326-1:2013	

Ilustración 1-1 (Pigeon)

DATOS TÉCNICOS



Entradas: 4 sondas de temperatura Pt1000, 1 sensor Grundfos Direct Sensors™, 2 entradas PWM con retroalimentación (CS Plus bidireccional)

Salidas: 1 relé semiconductor, 1 salida PWM (CS/2), 2 relés semiconductores, 1 salida PWM (CS/4), 2 relés semiconductores, 2 salidas PWM (CS Plus)

Frecuencia PWM: 512 Hz

Tensión PWM: 10,5 V

Potencia de salida:
1 (1) A 240 V~ (relé semiconductor)

Potencia total de salida:
1 A 240 V~ (CS/2)
2 A 240 V~ (CS/4, CS Plus)

Alimentación: 100 ... 240 V~ (50 ... 60 Hz)

Tipo de conexión: Y

Standby: 0,58 W (CS/2), 0,60 W (CS/4), 0,59 W (CS Plus), 0,61 W (CS Plus bidireccional)

Clases de controles de temperatura:
I (CS/4, CS Plus, CS Plus bidireccional)

Contribución a la eficiencia energética:
1 % (CS/4, CS Plus, CS Plus bidireccional)

Funcionamiento: tipo 1.C.Y

Ratio de sobretensión transitoria: 2,5 kV

Interfaz de datos: VBus® de RESOL

Transmisión de corriente VBus®: 35 mA

Funciones: función captador de tubos y función termostato (CS/4, CS Plus), control de funcionamiento, contador de horas de funcionamiento, control de velocidad y balance térmico

Carcasa: de plástico, PC-ABS y PMMA

Montaje: sobre pared o en cuadro de conexiones

Visualización / Pantalla: pantalla System Monitoring para visualizar el sistema con un campo de 16 segmentos y otro de 7, 8 símbolos para visualizar el estado del sistema

Manejo: con las 3 teclas frontales

Tipo de protección: IP 20 / IEC 60529

Categoría de protección: I

Temperatura ambiente: 0 ... 40 °C

Índice de contaminación: 2

Dimensiones: 172 x 110 x 46 mm

Ilustración 1-2 (Resol)

Ilustración 1-3 (Contazara)

	Clase de potencia		1.5-1	2.0-1	2.5-1	3.0-1	3.0-2	3.6-1	3.6-2	4.6-2
Lado de entrada (CC)	Potencia fotovoltaica máx. (cos φ = 1)	kWp	2,3	3,0	3,75	4,5		5,4		6,9
	Potencia CC nominal	kW	1,54	2,05	2,56	3,07		3,77		4,74
	Tensión de entrada nominal ($U_{CC,r}$)	V	350							
	Tensión de entrada de inicio ($U_{CCinicio}$)	V	100							
	Rango de tensión de entrada ($U_{CCmín}$ - $U_{CCmáx}$)	V	75-450	75-450	75-450	75-750		75-750		75-750
	Rango PMP con potencia nominal en el modo de un seguidor ($U_{PMPmín}$ - $U_{PMPmáx}$)	V	120-360	160-360	200-360	230-600		280-600		360-600
	Rango PMP con potencia nominal en el modo de dos seguidores ($U_{PMPmín}$ - $U_{PMPmáx}$)	V	-	-	-	-	115-600	-	140-600	180-600
	Rango de tensión de trabajo PMP ($U_{PMPtrabmín}$ - $U_{PMPtrabmáx}$)	V	75-360	75-360	75-360	125-600		150-600		150-600
	Tensión de trabajo máx. ($U_{CCtrabmáx}$)	V	450	450	450	750		750		750
	Corriente de entrada máx. ($I_{DCmáx}$) por entrada CC	A	13							
	Corriente de cortocircuito FV máx. (I_{SC_PV}) por entrada CC	A	15							
	Número de entradas CC		1	1	1	1	2	1	2	2
	Número de entradas CC bidireccionales		1	1	1	1	2	1	2	2
	Número de seguidores PMP Indep.		1	1	1	1	2	1	2	2
Lado de salida (CA)	Potencia nominal, cos φ = 1 ($P_{CA,r}$)	kW	1,5	2,0	2,5	3,0		3,7		4,6
	Potencia aparente de salida máx., cos Φ, $_{adj}$	kVA	1,5	2,0	2,5	3,0		3,7		4,6
	Tensión de salida mín. ($U_{CAmín}$)	V	185							
	Tensión de salida máx. ($U_{CAmáx}$)	V	276							
	Corriente de salida asignada ($I_{CA,r}$)	A	6,6	8,7	10,9	13,1		16		20
	Corriente de salida máx. ($I_{CAmáx}$)	A	12	12	14	14		16		20
	Corriente de cortocircuito (Peak/RMS)	A	21/12	21/12	24/12	24/16		27/16		20
	Conexión de red		1N~, 230V, 50 Hz							
	Frecuencia de referencia (f_r)	Hz	50 - 60							
	Frecuencia de red mín/máx ($f_{mín}$ / $f_{máx}$)	Hz	45...65							
	Margen de ajuste del factor de potencia (cos $\varphi_{CA,r}$)		0,8...1...0,8							
	Factor de potencia con potencia nominal (cos $\varphi_{CA,r}$)		1							
	Coeficiente de distorsión armónico máx.	%	<3							
	Espera/espera incl. medición del consumo doméstico las 24 h	W	<3,0/<20,0							
η	Coeficiente de rendimiento máx.	%	97,4	97,4	97,4	97,0		97,0		97,4
	Coeficiente europeo de rendimiento	%	96,1	96,5	96,6	96,3		96,3		96,9
	Coeficiente de rendimiento de adaptación PMP	%	>99,8							

Ilustración 1-4 (Inversor Kostal)

ABSOLUTE MAXIMUM RATINGS*

Voltage on any pin relative to ground	–0.5V to +6.0V
Operating temperature	–55°C to +100°C
Storage temperature	–55°C to +125°C
Soldering temperature	See J-STD-020A Specification

*These are stress ratings only and functional operation of the device at these or any other conditions above those indicated in the operation sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods of time may affect reliability.

DC ELECTRICAL CHARACTERISTICS (-55°C to +100°C; V_{PU}=3.0V to 5.5V)

PARAMETER	SYMBOL	CONDITION	MIN	TYP	MAX	UNITS	NOTES
Pullup Supply Voltage	V_{PU}		3.0		5.5	V	1,2
Thermometer Error	t_{ERR}	-10°C to +85°C			±½	°C	3
		-55°C to +100°C			±2		
Input Logic Low	V_{IL}		-0.3		+0.8	V	1,4,5
Input Logic High	V_{IH}		3.0		5.5	V	1,6
Sink Current	I_L	$V_{I/O}$=0.4V	4.0			mA	1
Active Current	I_{DQA}			1	1.5	mA	7
DQ Input Current	I_{DQ}			5		µA	8
Drift				±0.2		°C	9

Ilustración 1-1 (Sonda DS1820)

Anexo 2. Scripts y Diagramas de flujo.

Presentamos los códigos relativos a los scripts que controlan los componentes del proyecto, en todos ellos hay comentarios explicativos, y sus respectivos diagramas de flujo, en orden: Resol, Contazara, inversor Kostal, sonda DS1820.

CODIGO RELATIVO RESOL:

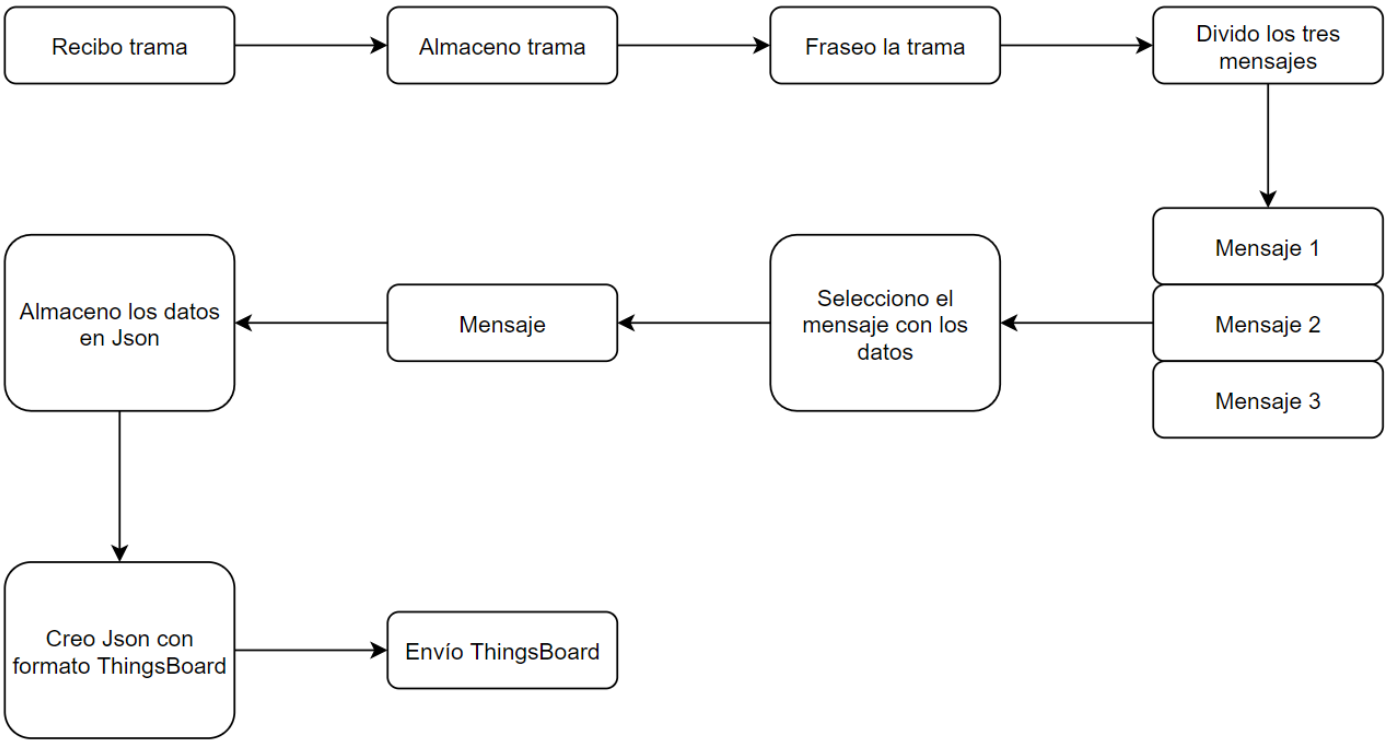


Ilustración 2-1 (Gestor de Resol)

```
import serial
import time
import paho.mqtt.client as mqtt
import Json

#Inicializo la conexión con ThingsBoard y la conexión en serie con el Resol

ser = serial.Serial('/dev/ttyACM0',9600)
client = mqtt.Client()
client.username_pw_set("jSmv4ESDzbPB1AljkvYF")
topic = "v1/devices/me/telemetry"
client.connect("tfg.howlab.es",41883)
ejecutar=1
client.publish(topic,'{"conectado":"1"}')


#Función para publicar Json
def publicar(telem):
   j=len(telem)
   for i in range(0,j):
      client.publish(topic, telem[i])
#Función para extraer la versión del protocolo del mensaje
```

```
 def get_protocolversion(msg):
    if len(msg) > 3:
        if hex(msg[4]) == '0x10': return "PV1"
        if hex(msg[4]) == '0x20': return "PV2"
        if hex(msg[4]) == '0x30': return "PV3"
    return "UNKNOWN"

#Funcion a la que le doy la trama y me devuelve los mensajes separados
def splitmsg(buf):
    return buf.split(bytes.fromhex('AA'))[1:-1]

#Leo datos hasta tener mínimo un mensaje

def readstream():
    data = recv()
    while data.count(bytes.fromhex('AA')) < 4:
        data += recv()
    return data

#Recibo bits de uno en uno.
def recv():
    dat = ser.read(1)
    #print(dat)
    return dat

# Le doy formato a un byte como hexadecimal
def format_byte(byte):
    return hex(ord(byte))[0:2] + '0' + hex(ord(byte))[2:] if len(hex(ord(byte))) < 4 else
hex(ord(byte))


# Devuelve en número de partes del mensaje
def get_frame_count(msg):
    return gb(msg, 7, 8)

# Devuelve el destino del mensaje
def get_destination(msg):
    return (msg[1]) + (msg[0])

#Devuelve los datos de una parte del mensaje
def integrate_septett(frame):
    data = ''
    septet = (frame[4])

    for j in range(4):
        if septet & (1 << j):
            data += chr((frame[j]) | 0x80)
        else:
            data += chr(frame[j])

    return data


# Devuelve el valor numérico de una serie de Bytes (complemento a dos)
def gb(data, begin, end):  # GetBytes
    wbg = sum([0xff << (i * 8) for i, b in enumerate(data[begin:end])])
    s = sum([(b) << (i * 8) for i, b in enumerate(data[begin:end])])

    if s >= wbg / 2:
        s = -1 * (wbg - s)
```

```
    return s

# extraer información del mensaje
def get_payload(msg):
    payload = ''
    for i in range(get_frame_count(msg)):
        payload += integrate_septett(msg[9+(i*6):15+(i*6)])
    return payload

while (ejecutar):
    #Leo el stream y separo los mensajes. Inicializo los diccionarios en los que usare
    #los Json
    mensaje = readstream()
    mensajeseparado = mensaje.split(bytes.fromhex('AA'))
    telemetria = dict()
    result = dict()
    packet = dict()
    DatosEnergia = dict()
    TelemetriaEnergia= dict()
    time.sleep(1)
    #abro los Json que voy a utilizar
    with open('/home/pi/paquete.json') as file:
        result = json.load(file)
    with open('/home/pi/Telemetria.json') as file:
        telemetria= json.load(file)
    with open('/home/pi/DatosEnergia.json') as file:
        DatosEnergia= json.load(file)
    with open('/home/pi/TelemetriaEnergia.json') as file:
        TelemetriaEnergia=json.load(file)
    packet = result['vbusSpecification']['packet']
    packet2= DatosEnergia['vbusSpecification']['packet']



    #Ahora repaso los mensajes buscando el que me interesa, lo hago sabiendo el protocolo y
    #destinatario que lleva.
    for i in range(0, 3):
        if get_protocolversion(mensajeseparado[i]) == 'PV1' and
.                                                   get_destination(mensajeseparado[i]) == 16:
            datos = get_payload(mensajeseparado[i])
            for j in range(0,16):
                inicio = int(packet[0]['field'][j]['offset'])
                tamaño = ((int(packet[0]['field'][j]['bitSize'])+1)/8)
                final = int(inicio + tamaño)
                factor = float(packet[0]['field'][j]['factor'])
                if (int(packet[0]['field'][j]['bitSize'])==1):
                    packet[0]['field'][j]['bits'] = datos[inicio]
                    bit1 = int(format_byte(packet[0]['field'][j]['bits'][0]), 0)
                    packet[0]['field'][j]['mesure'] = bit1

                if (tamaño== 1.0):
                    packet[0]['field'][j]['bits'] = datos[inicio:final]
                    bit1 = int(format_byte(packet[0]['field'][j]['bits'][0]),0)
                    packet[0]['field'][j]['mesure'] = bit1*factor
                if (tamaño == 2.0):
                    packet[0]['field'][j]['bits'] = datos[inicio:final]
                    bit1 = int(format_byte(packet[0]['field'][j]['bits'][0]),0)
                    bit2 = int(format_byte(packet[0]['field'][j]['bits'][1]),0)
                    packet[0]['field'][j]['mesure'] = bit1*factor+bit2*factor*256
```

```
            if (tamaño == 4.0):
                packet[0]['field'][j]['bits'] = datos[inicio:final]
                bit1 = int(format_byte(packet[0]['field'][j]['bits'][0]),0)
                bit2 = int(format_byte(packet[0]['field'][j]['bits'][1]),0)
                bit3 = int(format_byte(packet[0]['field'][j]['bits'][2]), 0)
                bit4 = int(format_byte(packet[0]['field'][j]['bits'][3]), 0)
                packet[0]['field'][j]['mesure'] = bit1*factor+bit2*factor*256+bit3*factor*65536+
.                                                 bit4*factor*16777216
            #Ahora voy rellenando el Json que enviare a ThingsBoard
            nombre = str(packet[0]['field'][j]['name'])
            dato = str(packet[0]['field'][j]['mesure'])
            telemetria[j] = '{"' + nombre + '":"' + dato + '"}'
            #Almaceno el dato de este sensor que corresponde al Acumulador para calcular
            #la energía en el scritp que ejecuta los demás.
            if nombre == ("Sensor Temperatura 2"):
                text_file = open("temperaturaColector.txt", "w")
                n = text_file.write(dato)
                text_file.close()
            #Publico en ThingsBoard el Json con la Telemetría
            publicar(telemetria)
        #Ahora hago lo mismo con el mensaje energético, esto solo es necesario en la
instalación real
        #Todos los pasos son semejantes a la sección anterior.
        if get_protocolversion(mensajeseparado[i]) == 'PV1' and
.                                                   get_destination(mensajeseparado[i])  ==  21:
            datos = get_payload(mensajeseparado[i])
            for j in range(0, 4):
                inicio = int(packet2[0]['field'][j]['offset'])
                tamaño = ((int(packet2[0]['field'][j]['bitSize']) + 1) / 8)
                final = int(inicio + tamaño)
                factor = float(packet2[0]['field'][j]['factor'])
                if (tamaño == 0.125):
                    packet2[0]['field'][j]['bits'] = datos[inicio]
                    bit1 = int(format_byte(packet2[0]['field'][j]['bits'][0]), 0)
                    packet2[0]['field'][j]['mesure'] = bit1
                if (tamaño== 1.0):
                packet2[0]['field'][j]['bits'] = datos[inicio:final]
                bit1 = int(format_byte(packet2[0]['field'][j]['bits'][0]),0)
                packet2[0]['field'][j]['mesure'] = bit1*factor
              if (tamaño == 2.0):
                packet2[0]['field'][j]['bits'] = datos[inicio:final]
                bit1 = int(format_byte(packet2[0]['field'][j]['bits'][0]),0)
                bit2 = int(format_byte(packet2[0]['field'][j]['bits'][1]),0)
                packet2[0]['field'][j]['mesure'] = bit1*factor+bit2*factor*256

              if (tamaño == 4.0):
                packet2[0]['field'][j]['bits'] = datos[inicio:final]
                bit1 = int(format_byte(packet2[0]['field'][j]['bits'][0]),0)
                bit2 = int(format_byte(packet2[0]['field'][j]['bits'][1]),0)
                bit3 = int(format_byte(packet2[0]['field'][j]['bits'][2]), 0)
                bit4 = int(format_byte(packet2[0]['field'][j]['bits'][3]), 0)
                packet2[0]['field'][j]['mesure'] =bit1*factor+bit2*factor*256+bit3*factor*65536+
.                                                 bit4*factor*16777216
              nombre = str(packet2[0]['field'][j]['name'])
              dato = str(packet2[0]['field'][j]['mesure'])
              TelemetriaEnergia[j]='{"' + nombre + '":"' + dato + '"}'
              publicar(TelemetriaEnergia)
              ejecutar = 0
```

CODIGO RELATIVO AL CONTAZARA:

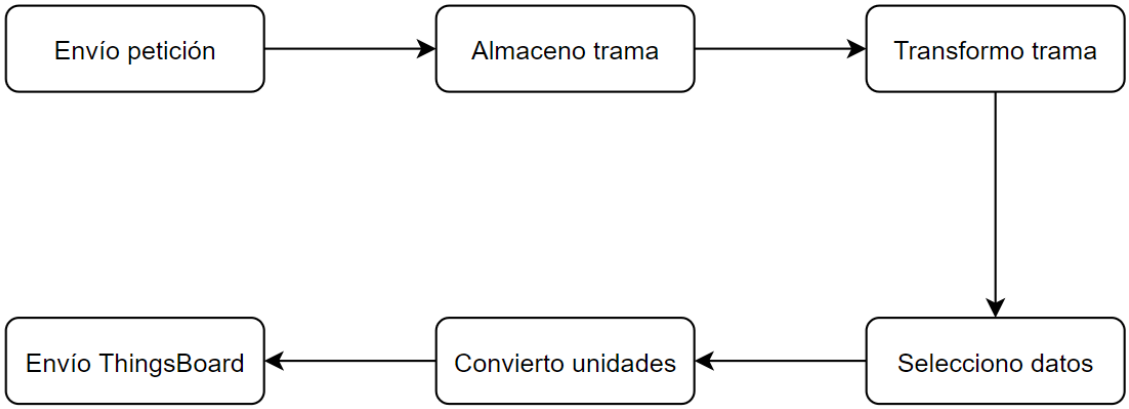


Ilustración 2-1 (Contazara)

```
import serial
import time
import paho.mqtt.client as mqtt
import RPi.GPIO as GPIO

#Inicializo la conexion con thingsboard
client = mqtt.Client()
client.username_pw_set("ipFLxyRyG5jzYPafjxk9")
topic = "v1/devices/me/telemetry"
client.connect("192.168.1.15")

#Inicializo los pines que activaran el envío de datos y la conexión serie.
GPIO.setmode(GPIO.BCM)
GPIO.setup(34, GPIO.OUT)
GPIO.setup(35, GPIO.OUT)
GPIO.output(34, False) #ENABLE active
GPIO.output(35, True)
contador = serial.Serial('/dev/ttyUSB0', 1200)
dat = 0
BytesIn = 0
recibido = 0
transformado = 0
preparado = 0
enviar = 0
BitEmpieza = 0
ejecutar=1
while ejecutar:
    #Creo el array donde almacenare los datos y hago la petición del envío de información(lógica negada)
    data = bytearray()
    BytesIn = 0
    GPIO.output(35,False)
    time.sleep(4)
    GPIO.output(35,True)
    time.sleep(4)
    while contador.inWaiting():
        time.sleep(5)
        while contador.inWaiting():
            #Leo la lectura del contador y añado un 0 al final
            dat = contador.read()
```

```
            data.append(int.from_bytes(dat, byteorder='big'))
            BytesIn = BytesIn + 1
            recibido = 1
        data.append(0)


if (recibido):
    #Transformo de 8  bit 1 stop a 7 bits 2 stop
    for i in range(0, BytesIn):
        data[i] = data[i] & 0b01111111
    recibido = 0
    transformado = 1

if (transformado):
    #Busco en la trama el la serie "\tRC" que indica el inicio de la zona donde esta la lectura
    BitEmpieza = data.find(b"\tRC")
    medida = bytearray(12)

    #Almaceno los 12 bytes que contienen la lectura.
    for i in range(0,12):
        medida[i] = data[BitEmpieza + i + 3]
    unidad = data[16]
    medidaInt=int(medida)
    #el carácter 16 son las unidades 1 = m ^ 3, 2 = litros, 3 = US galones, 4 = Imp gallones
if unidad == 1:
    #multiplico por el valor necesario para obtener litros
        medidaInt = medidaInt * 1000
    if unidad == 3:
        medidaInt = medidaInt * 3.78541
    if unidad == 4:
        medidaInt = medidaInt * 4.54609
    transformado = 0
    enviar = 1
if (enviar):
    #Envío a ThingsBoard
    mensaje='{"Medida Contador 1" : "' + str(medidaInt) + '"}'
    client.publish(topic, mensaje)
    print(mensaje)
    enviar=0
ejecutar=0
```

CODIGO RELATIVO AL INVERSOR:

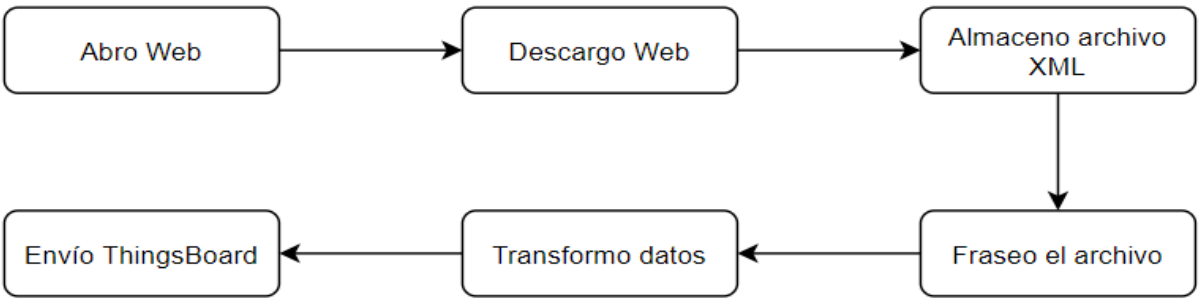


Ilustración 2-3 (Inversor Kostal)

```
import xml.etree.ElementTree as ET
import requests
import paho.mqtt.client as mqtt
import time

#Inicio conexión con ThingsBoard
client = mqtt.Client()
client.username_pw_set("jSmv4ESDzbPB1AljkvYF")
topic = "v1/devices/me/telemetry"
client.connect("tfg.howlab.es",41883)

#Descargo la pagina web y la almaceno en un archivo XML
telemetria=dict()
url = 'http://192.168.1.162/measurements.xml'
respuesta = requests.get(url)
with open('measurements.xml','wb') as file:
    file.write(respuesta.content)

root = ET.parse('measurements.xml').getroot()


#Almaceno el nombre y el valor de todas las variables del archivo
for i in range (0,13):
    value=str(root[0][0][i].get('Value'))
    nombre=str(root[0][0][i].get('Type'))
    #Almaceno el DC_voltage y DC_Current para calcular la potencia DC
    if (nombre== "DC_Voltage") :
        DC_Voltage=value
    if (nombre == "DC_Current"):
        if (value=='None'):
            print(value)
            DC_Current=0
        if (value!="None"):
            print(value)
            DC_Current=value
    mensaje='{"%s":"%s"}'%(nombre,value)
    #Publico cada dato
    client.publish(topic,mensaje)
    print(mensaje)
    time.sleep(1)
#Calculo la potencia DC y la publico.
DC_Power=float(DC_Voltage)*float(str(DC_Current))
mensaje='{"DC_Power":"%s"}'%DC_Power
client.publish(topic,mensaje)
```

```
#Almaceno la potencia en un archivo de texto para poder acceder a ella en
#el scritp que ejecuta el resto y calcular la energía.
text_file = open("/home/pi/Potencia.txt", "w")
n = text_file.write(str(DC_Power))
text_file.close()

time.sleep(1)
```

CODIGO RELATIVO A LA SONDA DS1820:

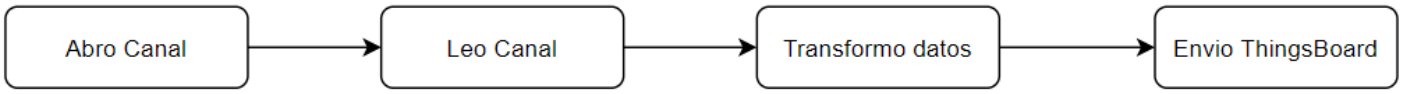


Ilustración 2-4 (Sonda DS1820)

```
import paho.mqtt.client as mqtt
from sys import stdout

# Inicio la comunicación con Thingsboard y la conexión con el sensor
id = "28.FFE130711603"
client = mqtt.Client()
client.username_pw_set("ipFLxyRyG5jzYPafjxk9")
topic = "v1/devices/me/telemetry"
client.connect("192.168.0.6")

try:
    # Leo la temperatura del sensor y la publico en ThingsBoard
    temp = open("/mnt/1wire/" + id + "/temperature", "r")
    t_raw = temp.read()
    temp.close()
    t = float(t_raw)
    mensaje = '{"Temperature":"%s"}' % t
    client.publish(topic, mensaje)
    stdout.flush()

except KeyboardInterrupt:
    break
```

CODIGO RELATIVO AL SCRIPT EJECUTOR:

```
import subprocess
import time
import paho.mqtt.client as mqtt

#Inicio la comunicación con Thingsboard, espero 30 s porque el script se va a ejecutar cuando la Pigeon se inicia
#y tengo que esperar a que se conecte a internet, de no hacerlo el script fallaría y no se iniciaría de manera
#automática el envió de datos.
time.sleep(30)
client = mqtt.Client()
client.username_pw_set("jSmv4ESDzbPB1AljkvYF")
topic = "v1/devices/me/telemetry"
client.connect("tfg.howlab.es",41883)


start = time.time()
temperaturaPrev=0
timeout=15

while True:
    try:
   #Ejecuto daca scritp que necesito utilizar, el timeout del Kostal es algo mayor porque el proceso tarda mas
   #en este script no ejecuto el código de Contazara ya que en la instalación real no hay contador conectado
        print ("Starting script1")
        subprocess.run("python3 /home/pi/Resol.py", shell=True,  timeout=timeout, check=True)
        print ("script1 ended")
        print ("Starting script2")
        subprocess.run("python3 /home/pi/SensorTemperatura.py",shell=True,  timeout=timeout, check=True)
        print ("script2 ended")
        print( "Starting script3")
        subprocess.run("python3 /home/pi/Kostal.py",shell=True, timeout=25, check=True)
        print("script3 ended")
   #Extraigo los datos necesarios para calcular la potencia del Colector y la generada.
        text_file = open('/home/pi/temperaturaColector.txt', "r")
        temperatura = text_file.read()
        text_file.close()
        pot = open('/home/pi/Potencia.txt',"r")
        potencia = pot.read()
        pot.close()
        ciclo = time.time()
        tiempociclo=(ciclo-start)
   #controlo el tiempo que ha pasado desde el ultimo calculo. Si ha pasado mas de un minuto lo realizo.
        if (tiempociclo>60):
            #Calculo la energía fotovoltaica con la potencia y la del acumulador con la diferencia de temperaturas
            EnergiaFoto=float(tiempociclo)*float(potencia)
            deltaTemperatura= float(temperatura)-float(temperaturaPrev)
            temperaturaPrev=temperatura
            EnergiaColector = deltaTemperatura*150*4186/3600000
            mensaje='{"Energia Colector":"%s"}' %EnergiaColector
            mensaje2='{"Energia Fotovoltaica":"%s"}' %EnergiaFoto
            client.connect("tfg.howlab.es",41883)
            client.publish(topic,mensaje)
            client.publish(topic,mensaje2)
            start=time.time()
```

```
        time.sleep(30)
    #Depuración de errores, si hay algún error en algún sript el programa sigue ejecutándose.
     except subprocess.TimeoutExpired:
        print('Tiempo Excedido')
     except subprocess.CalledProcessError:
        print('el proceso no termino correctamente')
```

CODIGO QUE EJECUTA EL SERVICIO QUE ARRANCA CON EL SISTEMA:

```
#!/bin/bash

python3 /home/pi/Ejecutar.py
```

Anexo 3. Json.

Presentamos los Json donde tenemos la información relativa a los datos del Resol. Por un lado, la parte general y por otro la parte energética, los demás Json que utilizamos parten en blanco y se rellenan en la ejecución del código.

JSON RESOL GENERAL:


```
{
 "vbusSpecification": {
  "device": [
   {
    "address": "0x1112",
    "mask": "0xFFFF",
    "name": "DeltaSol CS4 [Regler]",
    "isMaster": "true"
   },
   {
    "address": "0x1113",
    "mask": "0xFFFF",
    "name": "DeltaSol CS4 [Module]",
    "isMaster": "true"
   },
   {
    "address": "0x1121",
    "mask": "0xFFF0",
    "name": "DeltaSol CS4 [Heizkreis #]",
    "isMaster": "false"
   },
   {
    "address": "0x1131",
    "mask": "0xFFF0",
    "name": "DeltaSol CS4 [WMZ #]",
    "isMaster": "false"
   }
  ],
  "packet": [
   {
    "destination": "0x0010",
    "source": "0x1112",
    "command": "0x0100",
    "field": [
     {
      "offset": "0",
      "name": "Sistema",
      "bitSize": "31",
      "factor": "0.1",
      "unit": " °C"
     },
     {
      "offset": "4",
      "name": "Sensor Temperatura 1",
      "bitSize": "15",
      "factor": "0.1",
      "unit": " °C"
     },
     {
```

```
        "offset": "6",
        "name": "Sensor Temperatura 2",
        "bitSize": "15",
        "factor": "0.1",
        "unit": " °C"
      },
      {
        "offset": "8",
        "name": "Sensor Temperatura 3",
        "bitSize": "15",
        "factor": "0.1",
        "unit": " °C"
      {
        "offset": "10",
        "name": "Sensor Temperatura 4",
        "bitSize": "15",
        "factor": "0.1",
        "unit": " °C"
        },
        {
        "offset": "12",
        "name": "Sensor Temepratura 5",
        "bitSize": "7",
        "factor": "1",
        "unit": " %"
        },
        {
        "offset": "34",
        "name": "Velocidad bomba 1",
        "bitSize": "7",
        "factor": "1",
        "unit": " h"
        },
        {
        "offset": "35",
        "name": "Velocidad bomba 2",
        "bitSize": "7",
        "factor": "1",
        "unit": " %"
        },
{
        "offset": "36",
        "name":"Velocidad bomba 3",
        "bitSize": "7",
        "factor": "1",
        "unit": " %"
      },
      {
        "offset": "37",
        "name":"Velocidad bomba 4",
        "bitSize": "7",
        "factor": "1",
        "unit": " %"
      },
      {
        "offset": "48",
        "name":"Segundos de funcionamiento Bomba 1",
        "bitSize": "31",
        "factor": "1",
        "unit": " s"
```

```
        },
        {
         "offset": "52",
         "name":"Segundos de funcionamiento Bomba 2",
         "bitSize": "31",
         "factor": "1",
         "unit": "s"
        },
        {
         "offset": "56",
         "name":"Segundos de funcionamiento Bomba 3",
         "bitSize": "31",
         "factor": "1",
         "unit": "s"
        },

        {
         "offset": "40",
         "name":"Cantidad de Energia",
         "bitSize": "31",
         "factor": "1",
         "unit":"Wh"
         },
        {
         "offset": "88",
         "name": "AeroTermo",
         "bitSize": "1",
         "factor": "100"
        }


       ]
      }
    ]
  }
}
```

JSON RESOL ENERGIAS:


```
{
 "vbusSpecification": {
  "device": [
   {
    "address": "0x1112",
    "mask": "0xFFFF",
    "name": "DeltaSol CS4 [Regler]",
    "isMaster": "true"
   },
   {
    "address": "0x1113",
    "mask": "0xFFFF",
    "name": "DeltaSol CS4 [Module]",
    "isMaster": "true"
   },
   {
    "address": "0x1121",
    "mask": "0xFFF0",
    "name": "DeltaSol CS4 [Heizkreis #]",
    "isMaster": "false"
   },
   {
    "address": "0x1131",
    "mask": "0xFFF0",
    "name": "DeltaSol CS4 [WMZ #]",
    "isMaster": "false"
   }
  ],
  "packet": [
   {
    "destination": "0x0010",
    "source": "0x1112",
    "command": "0x0100",
    "field": [
     {
      "offset": "0",
      "name": "Whert",
      "bitSize": "31",
      "factor": "1",
      "unit": " Wh"
     },
     {
      "offset": "4",
      "name": "Poder",
      "bitSize": "31",
      "factor": "1",
      "unit": " °W"
     },
     {
      "offset": "8",
      "name": "Generado Hoy",
      "bitSize": "31",
      "factor": "1",
      "unit": " Wh"
     },
     {
      "offset": "12",
      "name": "Generado Semana",
```

```
          "bitSize": "31",
          "factor": "1",
          "unit": " Wh"
        }
      ]
     }
   ]
 }
}
```