Smart Home

By Richard & Sylvia Croxall version 0.05.

# Introduction

This document explains how to build an automated Smart Home using

1. This Documentation
2. A set of Rules that you write (using an example)
3. The Rules Compiler (Compiler2)
4. A Raspberry Pi (2 or later)
5. The Runtime Environment (Smart8r)
6. An X-10 Power Line Controller
7. X-10 Appliance modules, Lamp modules and an Infrared AV Controller.
8. A 433 MHz radio to X-10 Transceiver
9. 433 MHz PIR sensors
10. A Philips Hue Hub
11. Philips Hue Lamps

We include a sample set of rules written to suit our house.

# History

Around 1995, we purchased a set of X-10 modules and starting writing software for a PC to interface via the control lines on an RS-232 interface with a TW523 Power Line Controller which then sent messages over the mains to X-10 switches. This was written in ‘C’ and ran under MSDOS 6.22 on an old Amstrad PC. The rules were hard coded into the application. A rule change meant new coding in ‘C’ and a re-compilation.

Later on it could not run under Windows 95 or later versions because these Operating Systems blocked direct access to the RS-232 port.

Fast forward to today and the rules compiler is a separate application written in C# and runs on Windows-10. The runtime portion is written in C++ and runs on a Raspberry Pi (v2 or greater is preferred due to multiple CPUs). We now use a more modern XM10FL Power Line Controller instead of a TW523.

# Basic Concepts

# X-10 Concepts

The X-10 standard has been around for many decades. It allows X-10 controllers to switch appliance modules off and on. Lamp modules can be switched off, on and dimmed/brightened. There are 22 basic commands. Other commands have been added later, but are not widely supported. Up to 256 different devices can be addressed using a combination of a house code (A-P) and a device code (1-16).

The 22 commands are:

| Command | Description |
| --- | --- |
| 1-16 | Select device 1-16 on this house code. |
| On | Switch on any selected device on this house code. |
| Off | Switch off any selected device on this house code. |
| Dim | Dim by 1 level (17 levels) any selected lamp module on this house code. |
| Bright | Brighten by 1 level (of 17 levels) any selected lamp module on this house code. |
| All Lights On | Switch on all lamp modules on this house code |
| All Units Off | Switch off all modules (lamp or not) on this house code. |

All of the commands are sent as high frequency 120kHz tones over the mains wiring within a house.

X-10 messages are not acknowledged. Electrical noise or attenuation may prevent X-10 message from getting through. X-10 messages may sometimes clash causing messages to be lost.

Unfortunately with the advent of switched mode power supplies, many modern devices, with switched mode power supplies, create a lot of electrical noise on the mains circuits preventing X-10 signals propagating well. These errant devices with switched mode power supplies must be fitted with an FM10U X-10 Filter to allow reliable X-10 communication to take place.

The XM10FL is able to listen to its own transmissions. Where there is a clash with another transmitter, the Smart8r runtime software detects this and retransmits commands.

Some devices (e.g. our Sony TV) attenuate X-10 commands when they are switched on. The runtime software has to send the “All Units Off” command 3 times to ensure all devices receive the “All Units Off” command.

# Philips Hue Concepts

Philips Hue Lamps have the intelligence built into a lamp rather than the switch to the lamp. The lamps communicate using a 2.4GHz radio protocol known as Zigbee. The lamps are controlled by a device such as the Philips Hue Hub. Individual lamps are addressed by an EUI-64 MAC address of the form "00:17:88:01:03:12:12:36-0b". (MAC addresses are commonly EUI-48 MAC addresses which only use 6 pairs of hex digits whereas EUI-64 MAC addresses use 8 pairs of hex digits followed by “-0b”.) Initially, these EUI-64 MAC addresses can be obtained by interrogating the lamps using the Philips Hue Hub.

As the lamps do not run hot, for convenience, we have added Dymo labels to the bottom of each of our 6 lamps with the EUI-64 MAC address.

In our case, the Philips Hue Hub receives its instructions over Ethernet from the Raspberry Pi.

# Rule Concepts

# Introduction

Whereas other house control systems major on a person walking into the lounge and saying “Alexa lounge lights on”, our system uses a MS13E PIR sensor to detect a person entering, checks if it is dark and then switches the lights on automatically. It also switches the lights off again if no-one is detected for a specified number of minutes.

For convenience, the MS13E is battery powered and sends 433 MHz radio signals to a TM12U transceiver which forwards X-10 messages over the mains wiring.

The rules for our system are written, compiled and then used. Periodically requirements will change or become clearer and the rules will be re-written.

The rule language comprises a large number of declarations and then procedures which can be called.

# Declarations

# Comments

Comments have no meaning for the compiler but assist the author or later readers to understand the intent of the program. A comment starts with “//” and ends at the end of the line.

Example

// Copyright (C) 1995-2020 Richard Croxall - developer and architect\n

# Room Declaration

For convenient control of devices, they are grouped into rooms. Then all lights within a particular room can be turned on or off at the same time.

All devices must be designated as being in exactly one room. In our example we switch all of the Alexa and Google Home devices off and on as a single group. We have designated these as being in the virtual “Smart” room.

Example:

ROOM Conservatory, Lounge, DiningRoom, Kitchen, Hall, UtilityRoom, Landing, Bed1, Garage, Smart;

# House code Declaration

The X-10 system allows up to 16 house codes (named ‘A’ through to ‘P’). Each house code can contain up to 16 device codes (numbered 1 through to 16). Each house code declaration can nominate a Procedure to be called when an “All Units Off” or “All Lights On” command is sent for that house code.

Example:

HOUSECODE Downstairs B OFFPROCEDURE AllOffB ONPROCEDURE SomebodyIn;

# Device Declaration

Each physical device will have a corresponding device declaration. The X-10 devices can be:

* Lamp Device (device which can dim an incandescent bulb)
* Appliance Device (device which can only switch off or on)
* Appliance Lamp Device (device which can only be switched off or on, but logically should obey the “All Lights On” command. This is emulated in software.)
* Remote (one of 32 separate codes for an Infra-Red sender – used for switching on Sky receiver and tuning to channel 503.)
* Sensor (One of 2 separate codes for a PIR/darkness sensor)

Currently Philips Hue Devices can only be lamps.

Examples:

DEVICE APPLIANCELAMP Lounge.WallLights B 1 OFFPROCEDURE OffLoungeLEDLights ONPROCEDURE DoNothing;

DEVICE LAMP DiningRoom.CeilingLight B 4;

DEVICE SENSOR Conservatory.PIR P 1 OFFPROCEDURE DoNothing ONPROCEDURE SomebodyInConservatory;

DEVICE HUELAMP Lounge.MoodLight1 B "00:17:88:01:03:12:12:36-0b";

# Timeout Declaration

A timer is rather like an egg timer, something which can be started and after a designated time expires. The Timeout calls a Procedure when it expires.

Example:

TIMEOUT KitchenEmptyTimeout 00:30:00 OFFPROCEDURE NoOneInKitchen;

# Constant Declaration

The user can declare integer constants.

Example:

CONST daysInWeek = 7;

# Enum Declaration

The user can effectively declare a set of integer constants, each one greater than the previous. Enums do not have type checking.

Example:

ENUM personEnum (PersonAllOut, PersonSomeoneIn, PersonSomeoneInBed);

# Variable Declaration

Variables are global. No use case for local variables has been found yet.

A variable can either be a Boolean or an integer. A variable holds a single integer or a Boolean value.

Examples:

INT person = PersonAllOut;

BOOL SkyReceiverIsOn = FALSE;

# Procedure Declaration

A procedure is a named set of statements which can be called:

* From another procedure.
* From a Timeout.
* From a Timer.
* When a device is switched off or on.

Procedures currently cannot have parameters or local variables.

Example:

PROCEDURE RefreshDevices

REFRESHDEVICES;

END;

# Day Declaration

By default Saturday and Sunday are non-working days. By default Monday through to Friday are working days. Weekdays can be made into holidays for obvious holidays such as Christmas, Boxing Day and New Year.

Dates are entered in DD/MM/YY format.

British Summer Time (or Daylight Savings Time) occurs at a different day every year. This can be declared for forthcoming days in a Day declaration. This information is used to adjust local times to the GMT clock. Currently only BST and GMT are supported.

Typically Bank holidays for the next two or three years should be declared.

Examples:

DAY 25/12/20 HOLIDAY; // Christmas day

DAY 26/12/20 HOLIDAY; // Boxing Day

DAY 29/3/20 BST; // first day of BST for year

DAY 25/10/20 GMT; // day after last day of BST for year

# Timer Declaration

A timer is used for a sequence of related events. An event can be timed to happen **before** the timer fires. This can be useful where we want the television to be turned off automatically early on the night before a working day (school night) taking into account bank holidays.

Times of day and durations are entered as HH:MM or HH:MM:SS.

Each Timer consists of a series of Sequences. The Sequences have a firing time and days on which they should fire. E.g. “ALL”, “MON”, “TUE”, etc.

Each Sequence consists of a series of Events. In turn, each Event consists of a signed time offset and a Procedure name to call.

Example:

TIMER "Xmas";

SEQUENCE "jollymorning" 07:00 ALL;

EVENT 00:00 XmasLightsOn;

SEQUENCE "savepowerovernight" 23:00 ALL;

EVENT 00:00 XmasLightsOff;

END;

# Statements

A statement can be one of the following:

# If statement

This allows the result of a Boolean expression to be used to optionally execute some statements. It comprises an “IF”, a Boolean expression, a “THEN”, some statements and finally an “ENDIF”. Optionally the “ENDIF” can be preceded by an “ELSE” and more statements.

Examples:

IF person == PersonSomeoneIn THEN

IF SomeoneIsInKitchen THEN

SETDEVICE Kitchen.Amplifier ON;

SETDEVICE Kitchen.Tv ON;

ENDIF;

ELSE // when PersonSomeoneInBed do not switch Kitchen on

SETDEVICE Kitchen.Amplifier OFF;

SETDEVICE Kitchen.Tv OFF;

ENDIF;

# Call statement

Allows a procedure to be called **without** parameters.

Example:

CALL TvLights;

# Assignment statement

Allows the result of an expression to be assigned to a Boolean or integer global variable.

Example:

darkness = Dark;

# Increment or decrement statement

Allows the value of an integer global variable to be increased or decreased by 1.

Examples:

darkness++;

darkness--;

# Set device statement

Allows a previously declared device to be set “On”, “Off” or (for lamps) to one of 17 different levels of dimmed. The action can be delayed for a time (specified in hours, minutes & seconds) to allow someone to exit a room. The Set device statement can also specify a duration (specified in hours, minutes & seconds) that the device should remain switched on. If no duration is quoted the default is 12 hours.

The set device statement also allows the colour and colour loop to be set for Philips Hue Lamp Devices.

Example:

SETDEVICE Lounge.SkyReceiver ON;

SETDEVICE Lounge.MoodLight6 DIM12 Teal COLOURLOOP;

SETDEVICE Smart.GoogleAlexa OFF DELAYED 00:10:00;

SETDEVICE Bed1.ComputerLight ON DELAYED 00:01:00 DURATION 01:00:00;

# Refresh devices statement

Causes the runtime system to assume that each device could be in the wrong state and send the wanted state to each device again. In our sample code a Timer calls a Procedure to do this once per hour. Some X-10 switches seem to react to electrical noise and switch on randomly occasionally. Philips Hue devices seem to power up in the on-state after a power cut.

Example:

REFRESHDEVICES;

# Resynch clock statement

Causes the X-10 clock to be re-synchronised with the Raspberry Pi runtime clock. The X-10 clock is in turn synchronised by the Raspberry Pi to an internet time serving service. In our sample code a Timer calls a Procedure to do this once per day.

Example:

RESYNCHCLOCK;

# Reset Timeout statement

This statement causes a Timeout to be reset. When a Timeout expires (typically after a few minutes) a Procedure is called as defined in the Timeout declaration. This Procedure typically turns off lights when someone leaves a room.

Optionally, a Duration can be specified in the Reset statement which overrides the default Duration declared in the Timeout statement.

Example:

RESET LoungeEmptyTimeout DURATION 01:30:00;

# Compiler2 Architecture

This is the second major version of the rules compiler. So the Github folder, project and the executable are all called Compiler2.

The smart rules compiler reads a file of rules and translates them into a smart rules file for the runtime system. The syntax of the rules file is documented separately.

# Compiler2 Architecture

The compiler is written in C# and runs on Windows-10. The compiler comprises the following parts:

* 1. A Lexical Analyser
  2. A Syntax Analyser
  3. A Semantic Analyser
  4. A Code Generator

The compiler syntax analyser uses recursive descent to allow the analysis and generation of code for recursively defined statements and expressions. The compiler maintains a list of symbols which can be used to resynchronise the compiler with the input stream in the event of missing, extra or incorrect symbols. In order to allow the calling of procedures before they are declared, the compiler makes two passes of the source code. Errors for missing procedures are suppressed on the 1st pass in case they are declared after the call. Most errors are reported only on the 2nd pass of the code.

The compiler generates code for a stack-based virtual machine.

# Compiler2 options

The compiler can be invoked from the DOS command line. It takes the following parameters:

<filename> input file name or path

-o <filename> generated code output file or path

For example:

compiler2 smart.txt -o smartdummy.smt

# Use of Compiler2



# Smart8r Runtime Architecture

# Introduction

This is the eighth major version of the runtime system. Originally rules compilation and the runtime were a single program. When these were split, the runtime became “smart8r”. So the Github folder, project and the executable are all called “smart8r”.

The runtime code can either run under Windows or on a Raspberry Pi under Linux.

The code is written in C++. It contains code from another Github project HuePlusPlus to control the Philips Hue Lamps. The include paths have been altered to make it easier to compile on Windows or Linux. In turn HuePlusPlus contains another Github project for constructing and de-constructing JSON messages.

The code comprises 3 threads:

* Main Rules checking thread
* X-10 interface thread
* Philips Hue interface thread

On Windows, the X-10 interface thread has a very simple implementation sufficient only to check the rules run okay.

# Start-up

When the runtime starts, possibly after a power cut or a malfunction, it does not know what state any of the devices are in or what state they should be in. It runs the rules for the previous day to ascertain what state devices should be in for the time of day, day of the week and whether it is a holiday or weekend.

Then the runtime sends out messages to each declared device to put it into the wanted state. X-10 devices normally will revert to the off state after a power cut, but Philips Hue Lamps can default to the on-state.

# Windows

The Windows-10 version compiles under Visual Studio. Since it cannot talk to a physical XM10FL, the device is simulated in a simple minded way sufficient to allow the rules to be debugged. This means that no real X-10 devices are affected by the Windows version of the runtime system.

The HuePlusPlus provides a windows version of the Ethernet handler. This allows the runtime system to control the Philips Hue Lights exactly as they do on the Linux version of the runtime system.

# Compilation with Visual Studio 2019

If you get an error due to warning C4146, this article explains to turn off SDL under Project / Properties / C/C++ / General / SDL Checks/.

<https://stackoverflow.com/questions/37546340/msvs-2015-express-error-c4146-unary-minus-operator-applied-to-unsigned-type>

# Linux

The Debian Linux on the Raspberry Pi, is not really designed as a real-time operating system. To make the runtime perform acceptably well, it has to be set to run at a higher priority using a script called “start”. The runtime code also has to run as root to provide access to the GPIO ports.

After initially deducing the wanted states of each device and sending an update message to each device, the system becomes quiescent. This only changes when:

* A message is received externally from an X-10 Maxi-Controller
* A message is received from a PIR sensor
* The real-time clock causes a rule to fire.

The firing of rules may result in procedures being executed and updating the wanted states of devices. Finally where the wanted state of a device is different to the actual state of a device, messages are sent to update the device.

X-10 messages are very slow, so the X-10 messages sent are heavily optimised to minimise the send time. For example X-10 messages to do “All Lights On” or “All Units Off” are much quicker than sending individual device messages. Also where possible, several devices are selected at the same time to send “On”, “Off”, “Dim” or “Bright” messages.

X-10 messages are sent using a separate thread to act as an UART so as to send the individual bits (actually half bits) required to send a simple message.





# Automatic start after Reboot

When power is restored after a power failure the Raspberry Pi will restart automatically. However the standard Raspberry Pi does not have a real time clock. This can be addressed by:

* Adding a hardware real time clock with battery backup.
* Letting the Raspberry Pi obtain the time using the Ethernet connection.

The Smart8r program must also be automatically re-started. When using the Internet, the real time will not available for several minutes until the house router has rebooted and the NTP service interrogated.

This enhancement will be added in the near future.

# Keyboard Input

In order to allow some debugging activity on the Raspberry Pi, the runtime system accepts a small number of single key debugging commands. These are listed below:

‘a’: Advance the clock to the next timer firing.

‘b’: Set the active house code to ‘B’.

‘c’: Set the active house code to ‘C’.

‘d’: Print the current states of all declared devices.

‘f’: Send an off message with the current house code and device number.

‘h’: Advance the internal clock by 1 hour.

‘l’: Print the current states of all of the timers.

‘m’: Advance the internal clock by 1 minute.

‘n’: Set an on message with the current house code and device number.

‘p’: Set the active house code to ‘P’.

‘q’: Quit the program. Stops threads and prints “Goodbye cruel world”.

‘r’: Send messages to refresh all devices to their wanted states.

‘s’: Resynch the X-10 clock to the host Raspberry Pi’s clock.

‘t’: Print the time from the X-10 internal clock.

‘v’: Print the current values of all the variables.

‘0’ .. ‘9’: Set the current device number to that digit. If ‘1’ is used following by say ‘3’, the current device number is set to 13.

Other keys: Print “unknown key”.

# Thread Structure

# Introduction

The runtime system comprises 3 threads.

* Main Rules checking thread
* X-10 interface thread
* Philips Hue interface thread

The main thread initialises the system and starts the 2 other threads. The main thread runs the rules to change the wanted state of the devices. The main thread communicates with each of the other two threads with a message queue and a response queue. It sends a single command to the queue to change the actual device state and waits for the queue to empty before sending another one. Any responses from the threads are used to update the main thread’s knowledge of the state of devices.

# Main Thread

The main thread initialises, starts the other 2 threads and then enters an infinite loop. The loop only exits if the user presses ‘q’. Inside the loop the main thread runs the following steps:

* Check for any command keys pressed and execute them.
* Process any responses in the X-10 Thread Response Queue.
* Process any responses in the Philips Hue Thread Response Queue.
* Check for any X-10 devices not in their wanted state and generate messages in the X-10 thread message queue.
* Check for any Philips Hue devices not in the wanted state and generate messages to the Philips Hue thread message queue.
* Check for any Timer ready to fire and if ready, fire it and reschedule it.
* Check for any devices ready to switch “Off” or “On”.
* Check for any Timeout ready to switch off.
* If nothing happens, sleep for a second.

Many of the steps above can change the wanted state of devices which then cause messages to be sent.

# Sunset & Sunrise

“Sunset” and “Sunrise” are built in keywords. The program currently contains hardcoded times for sunrise and sunset in Poole UK every 2 weeks for the year. Values are interpolated for days between the stored values.

# Daylight Savings Time

In many countries in the world there is a convention to change the time by one hour between the summer and the winter. The date of the change varies by country. You must declare the dates of the first day of summer time and the first day of winter time using the “DAY” declarations.

Internally the runtime uses UTC (or GMT). When the user declares he wants the radio to come on at 08:00:00, he is referring to local time (whether that is summer or winter). Consequently local times (which in the UK might be in GMT or BST) are converted to UTC using declarations for the start (BST) and end (GMT) of Daylight Savings Time.

Currently only BST and GMT are supported.

# X-10 Thread

Messages sent to the X-10 devices are sent serially as high frequency 120kHz tones over the mains. Tones are only sent when the mains voltage crosses zero volts (100 times per second) so that there will be minimum electrical noise. A single bit is sent as either:

* 1 = 120kHz Tone followed by silence
* 0 = Silence followed by a 120kHz tone

So sending a single bit as two “half bits” takes 1/50th of a second. Each message requires

* 4 half bit ‘1110’ start synchronisation (2 bit times)
* 4 bit house code
* 1 bit command/device select switch
* 4 bit device number or command number
* 3 bit silence (duration 6 half bit times)

So each message requires 14 bit times. To provide some resistance to noise, each message is sent twice, so takes 28 bit times. To switch on device 10 on house code ‘B’ requires two messages:

1. Select device 10 on house code ‘B’.
2. Switch on all selected devices on house code ‘B’. (This finally deselects devices).

So switching on device 10 requires 56 bit times which at 50Hz is slightly more than 1 second.

As this takes such a long time, the main thread heavily optimises the messages sent.

Whilst the messages are being sent, the X-10 thread listens to the message being sent. If the received message is not identical to the message sent, the thread assumes another device is sending, so it waits several seconds and tries resending.

# Philips Hue Thread

The Philips Hue Thread sits between the main thread and the Philips Hue Hub. The Philips Hue Thread reads messages on the queue and sends them using the HuePlusPlus library to the Philips Hue Hub. Where it has several messages for the same lamp (e.g. on/off/dim level, colour and colour loop), they are optimised into a single message being sent to the Philips Hue Hub. The hub in turn passes the message to the lamp over Zigbee.

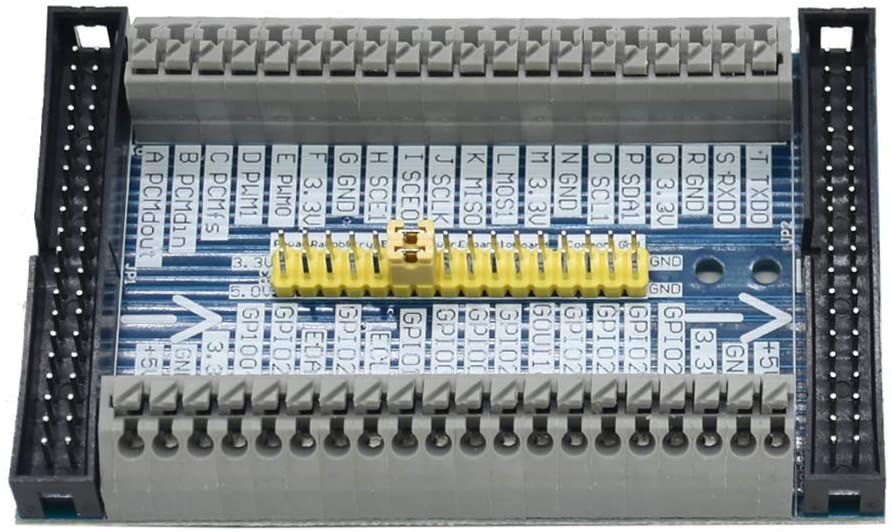
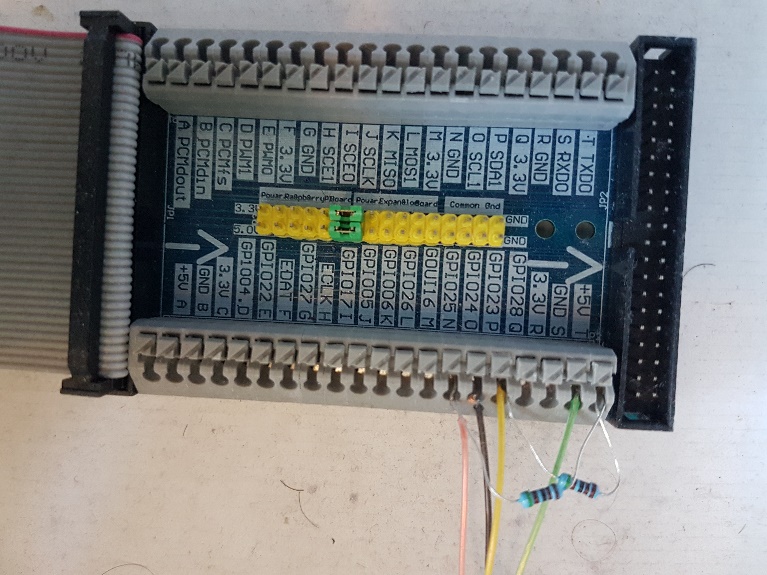
Any response from the lamp is returned via the hub and put into the response queue to the main thread.

# Hardware

# XM10FL

This is better described here: <https://www.uk-automation.co.uk/content/pdf/xm10man.pdf> .

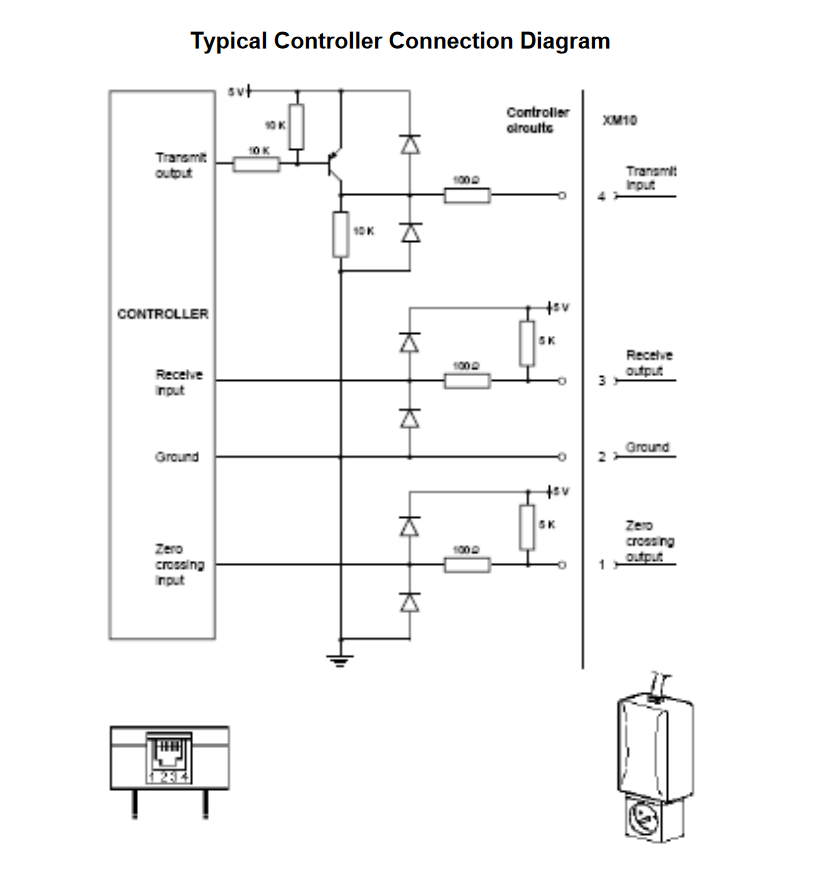
# Interface

The Raspberry Pi needs the GPIO bus to be connected to the XM10FL. The simplest way we found to do this was using an “ICQUANZX GPIO Cable Adapter+Raspberry Pi 2/3 Model B Multifunctional

Cascade Expansion Extension GPIO Board Module For Orange Pi PC”.

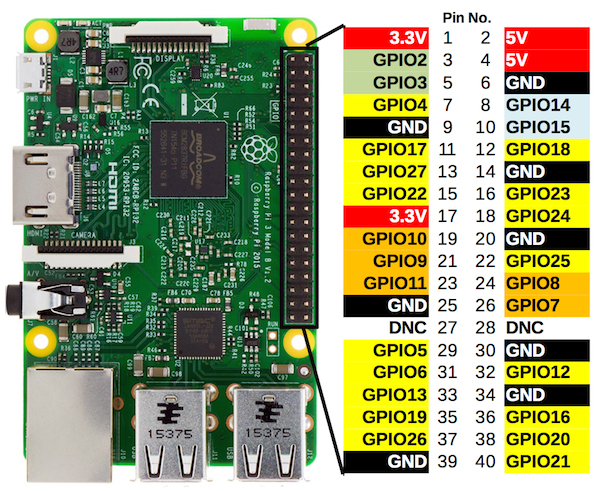
The XM10FL provides two inputs and one output:

1. Mains zero crossing via opto-isolator (requires pull up resistor)
2. 120kHz Tone being received via opto-isolator (requires pull up resistor)
3. Send 120kHz Tone over the mains via opto-isolator.



XM10FL interface circuit diagram between the Raspberry Pi (controller) and the XM10FL.







XM10FL interface. The mains is connected via a cable attached to the socket shown. The 4 core data cable is attached via another cable.

TODO: add circuit diagram showing GPIO pins, resistors and connections to XM10FL

# Transferring and running the Runtime code and the Rules

1. Write rules in notepad or your favourite text editor.
2. Compile the rules with the rules compiler.
3. Correct any errors and return to step 1.
4. Use batch file ‘copysmart.bat’ to copy files to a USB memory stick.
5. Start the Raspberry Pi.
6. Type command “sudo bash”.
7. Create a directory called “smart8r”.
8. Mount the USB memory stick.
9. Copy the files from the USB memory stick to the new folder.
10. Type “make”.
11. If there are no compilation errors, type “./start” to start an instance of the runtime at a high priority. (The “./” prefix explicitly allows an executable program in the current folder to be executed.)

# Rules Tutorial

# Requirements

This tutorial takes you through the process of writing the rules for a simple home automation system.

Let’s assume that you have the following requirements:

* You have 2 rooms to automate, Lounge and Bedroom
* You have the following devices in your Lounge:
  + TV
  + Light
* You have the following devices in your Bedroom:
  + Radio
  + Light
* On working days (Weekdays excluding bank holidays), you want the radio to come on in the morning from 07:30 until 08:30.
* On Weekends and bank holidays, you want the radio to come on at 09:00 until 10:00.
* If you go into the lounge from 07:30 until 08:30 on a working day, you want the TV to switch on to show the news.
* If you enter either the lounge or the bedroom when it is dark, you want the light to come on for 30 minutes (except when you are asleep in bed).
* You purchase and install
  + 2 off MS13E X-10 PIR sensors
  + 1 off TM12U X-10 “radio transceiver”
  + 2 off AM12U X-10 Appliance modules
  + 2 off LM12U X-10 Lamp modules (for table lamps) or plug replacement wall switches
  + 2 off SC503 X-10 Maxi controllers (no longer available).
  + 1 off XM10FL
  + 1 Raspberry Pi (v2 or later)

# Rules

# Rooms

Declare your two rooms.

ROOM Lounge, Bedroom;

# House Codes

Let’s split your rooms into different house codes to allow for future expansion.

HOUSECODE downstairs B OFFPROCEDURE AllOffB ONPROCEDURE SomebodyIn;

HOUSECODE upstairs C OFFPROCEDURE AllOffC ONPROCEDURE SomebodyIn;

HOUSECODE PIRS P OFFPROCEDURE DoNothing ONPROCEDURE DoNothing;Devices

Declare all of your devices with house codes reflecting which rooms they are in.

DEVICE APPLIANCELAMP Lounge.CeilingLights B 2;

DEVICE APPLIANCE Lounge.Tv B 15 OFFPROCEDURE TvOff ONPROCEDURE TvOn;

DEVICE APPLIANCELAMP Bedroom.CeilingLight C 1;

DEVICE APPLIANCE Bedroom.Radio C 13;

DEVICE SENSOR Lounge.PIR P 3 OFFPROCEDURE DoNothing ONPROCEDURE SomebodyInLounge;

DEVICE SENSOR Lounge.PIRDark P 4 OFFPROCEDURE DoNothing ONPROCEDURE GoingDark;

DEVICE SENSOR Bedroom.PIR P 11 OFFPROCEDURE DoNothing ONPROCEDURE SomebodyInBedroom;

DEVICE SENSOR Bedroom.PIRDark P 12 OFFPROCEDURE DoNothing ONPROCEDURE GoingDark;

# Timeouts

TIMEOUT LoungeEmptyTimeout 00:30:00 OFFPROCEDURE NoOneInLounge;

TIMEOUT Bedroom1EmptyTimeout 00:30:00 OFFPROCEDURE NoOneinBedroom1;

# Variables

ENUM personEnum (PersonAllOut, PersonSomeoneIn, PersonSomeoneInBed);

INT person = PersonAllOut;

ENUM DarknessEnum (Light, NearlyDark, Dark);

INT darkness = Dark;

ENUM TimeEnum (DayTime, BedTime, EncourageToBed, SleepTime, WakingUpTime);

INT whatTime = DayTime;

BOOL tvNewsWantedInLounge = false;

BOOL SomeoneIsInLounge = false;

BOOL SomeoneIsInBedRoom = false;

# Procedures

# DoNothing

PROCEDURE DoNothing

END;

# AllOffB

PROCEDURE AllOffB

IF whatTime == WakingUpTime THEN

CALL AllGoOut;

ELSE

IF whatTime == DayTime THEN

CALL AllGoOut;

ENDIF;

ENDIF;

END;

# AllOffC

PROCEDURE AllOffC // what happens when the C house code (upstairs) all off button is pressed

IF whatTime == WakingUpTime THEN

CALL AllGoOut;

ELSE

IF whatTime == DayTime THEN

CALL AllGoOut;

ELSE

whatTime = EncourageToBed;

whatTime = SleepTime;

ENDIF;

ENDIF;

END;

# SomebodyIn

PROCEDURE SomebodyIn

IF person == PersonAllOut THEN

person = PersonSomeoneIn;

ENDIF;

END;

# GoingDark

PROCEDURE GoingDark //Fires if any PIR says its Dark - However this also happens whenever we switch lights off.

IF darkness == NearlyDark THEN //check whether it is nearly time for sunset

CALL SunSet;

ENDIF;

END;

# SomebodyInLounge

PROCEDURE SomebodyInLounge

IF tvNewsWantedInLounge THEN

CALL TvOn;

ENDIF;

SomeoneIsInLounge = TRUE;

IF Lounge.Tv == ON THEN

RESET LoungeEmptyTimeout DURATION 01:30:00;

ELSE

RESET LoungeEmptyTimeout;

ENDIF;

CALL SomebodyIn;

IF darkness == Dark AND Lounge.Tv == OFF THEN

SETDEVICE Lounge.CeilingLights ON DURATION 00:20:00;

ENDIF;

END;

# SomebodyInBedroom

PROCEDURE SomebodyInBedroom

RESET Bedroom1EmptyTimeout;

CALL SomebodyIn;

IF darkness == Dark AND person != PersonSomeoneInBed THEN

SETDEVICE Bedroom.CeilingLight ON DURATION 00:20:00;

ENDIF;

END;

# TvOn

PROCEDURE TvOn

SETDEVICE Lounge.Tv ON DURATION 00:40:00;

END;

# WakeUpWorking

PROCEDURE WakeUpWorking

tvNewsWantedInLounge = true;

SETDEVICE Bedroom.Radio ON DURATION 01:01:00;

END;

# WakeUpNonWorking

PROCEDURE WakeUpNonWorking

SETDEVICE Bedroom.Radio ON DURATION 01:01:00;

END;

# NoOneInLounge

PROCEDURE NoOneInLounge

SomeoneIsInLounge = FALSE;

SETDEVICE Lounge.LAMP OFF;

END;

# NoOneInBedroom1

PROCEDURE NoOneinBedroom1

SETDEVICE Bedroom.LAMP OFF;

END;

# AllGoOut

PROCEDURE AllGoOut

tvNewsWantedInLounge = false;

SETDEVICE Lounge.APPLIANCE OFF;

SETDEVICE Lounge.LAMP OFF;

SETDEVICE Bedroom.APPLIANCE OFF;

SETDEVICE Bedroom.LAMP OFF;

END;

# SunRise

PROCEDURE SunRise // predefined action list that will fire automatically

darkness = Light;

SETDEVICE Lounge.LAMP OFF;

SETDEVICE Bedroom.LAMP OFF;

END;

# SunSet

PROCEDURE SunSet // predefined action list that will fire automatically

darkness = Dark;

IF SomeoneIsInLounge THEN

SETDEVICE Lounge.LAMP ON DURATION 01:00:00;

ENDIF;

IF SomeoneIsInBedRoom THEN

SETDEVICE Bedroom.LAMP ON DURATION 01:00:00;

ENDIF;

END;

# NearlyDarkAction

PROCEDURE NearlyDarkAction //Fires a little while before Sunset

darkness = NearlyDark;

END;

# RefreshDevicesAndResynchClock

PROCEDURE RefreshDevicesAndResynchClock

RESYNCHCLOCK;

REFRESHDEVICES;

END;

# RefreshDevices

PROCEDURE RefreshDevices

REFRESHDEVICES;

END;

# Day Declarations

DAY 29/3/20 BST; // first day of BST for year

DAY 25/10/20 GMT; // day after last day of BST for year

DAY 1/1/20 HOLIDAY; // New Year's Day Holiday

DAY 10/04/20 HOLIDAY; // Good Friday

DAY 13/04/20 HOLIDAY; // Easter Monday

DAY 8/5/20 HOLIDAY; // May Day Bank Holiday

DAY 25/5/20 HOLIDAY; // Spring Bank Holiday

DAY 31/8/20 HOLIDAY; // August Bank Holiday

DAY 25/12/20 HOLIDAY; // Christmas day

DAY 26/12/20 HOLIDAY; // Boxing Day

# Timer Declarations

# Wakeup Timer

TIMER "wakeupTimer";

SEQUENCE "morning" 07:30 WORKING;

EVENT 00:00 WakeUpWorking;

EVENT 01:02 AllGoOut;

SEQUENCE "lie-in" 09:00 NONWORKING;

EVENT 00:00 WakeUpNonWorking;

EVENT 01:02 AllGoOut;

END;

# RefreshDevices

TIMER "refreshdevices";

SEQUENCE "refreshingdevices" 00:30 ALL;

EVENT 00:00 RefreshDevices;

EVENT 01:00 RefreshDevices;

EVENT 02:00 RefreshDevicesAndResynchClock;

EVENT 03:00 RefreshDevices;

EVENT 04:00 RefreshDevices;

EVENT 05:00 RefreshDevices;

EVENT 06:00 RefreshDevices;

EVENT 07:00 RefreshDevices;

EVENT 08:00 RefreshDevices;

EVENT 09:00 RefreshDevices;

EVENT 10:00 RefreshDevices;

EVENT 11:00 RefreshDevices;

EVENT 12:00 RefreshDevices;

EVENT 13:00 RefreshDevices;

EVENT 14:00 RefreshDevices;

EVENT 15:00 RefreshDevices;

EVENT 16:00 RefreshDevices;

EVENT 17:00 RefreshDevices;

EVENT 18:00 RefreshDevices;

EVENT 19:00 RefreshDevices;

EVENT 20:00 RefreshDevices;

EVENT 21:00 RefreshDevices;

EVENT 22:00 RefreshDevices;

EVENT 23:00 RefreshDevices;

END;

# Darkness

TIMER "Darkness";

SEQUENCE "sunset" SUNSET ALL;

EVENT -01:00 NearlyDarkAction;

EVENT 00:00 SunSet;

SEQUENCE "sunrise" SUNRISE ALL;

EVENT 00:00 SunRise;

END;