Appendix 4: Raspberry Pi Driving Hardware using GPIO Pins

Project 4: Using Raspberry Pi General Purpose IO pins

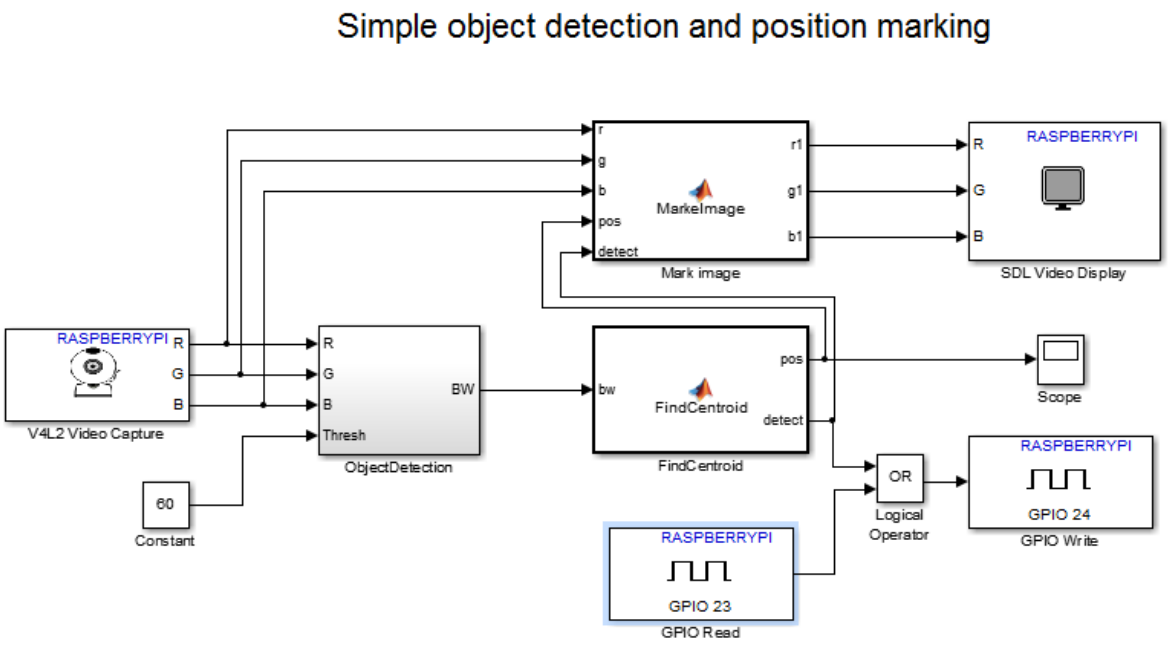
P4.1 Optional: Using General Purpose IO Pins via Simulink Blocks

Objective: To understand how to interface with hardware using GPIO Pins

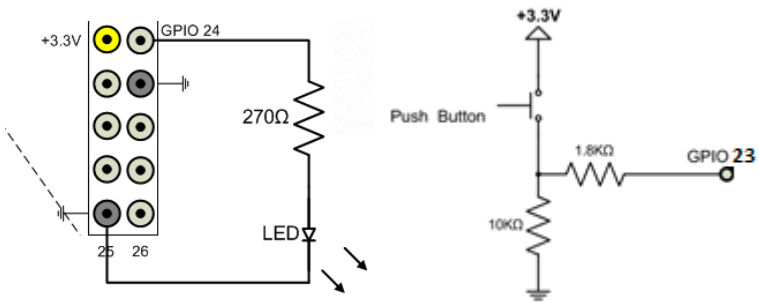
Task/ Challenge: Build a Simulink application which turns an output LED on as soon as it detects a green object. Take inputs from a push button to test the LED

Steps/ Approach:

1. It is possible to drive General Purpose IO pins on the Raspberry Pi using Simulink Blocks. Starting from the **ObjectMarker.slx** example, we can drive a LED every time we detect a green object by building the model as below



2. Use the following circuits to drive the LED and as an input of the Raspberry Pi



These simple circuits are realised on the breadboards. Using the above model you can see how the input can turn the LED on as well as detecting the green object.

3. Build and run the application and check that the LED does come on on detecting a green object or by pushing the push button.

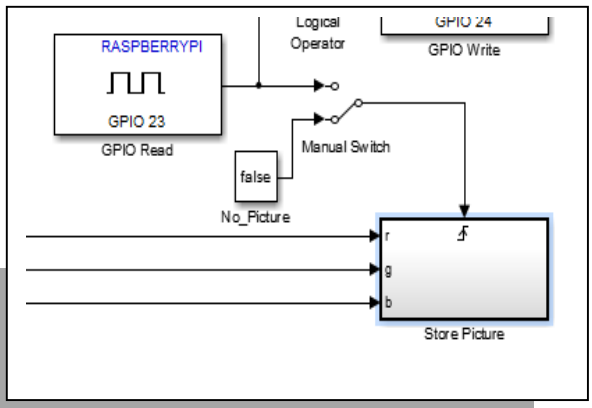
P4.2 Optional: Calling C Functions from Simulink

Objective: To demonstrate how to include calls to standard C libraries

Task/ Challenge: To run and examine a Simulink model which stores a picture for examination with MATLAB for further analysis

Steps/ Approach:

1. As a second example, load now the model **ObjectDetectionPosMarkStore.slx**. The store block captures the RGB signals every time the push button is pressed. It writes them into an ASCII files imgX.dat where X is determined programmatically by the number of times the pushbutton is pressed.



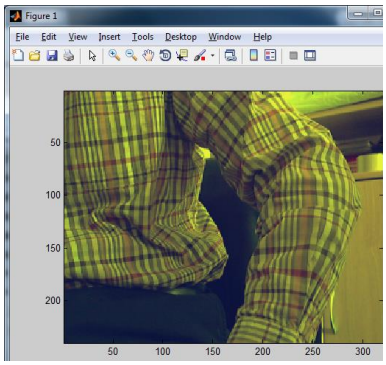
The code calls “C” functions sprintf & fprintf

```
function fcn(r,g,b, counter)
%#codegen
% Save 320x240 colour image to a file
format = ['%d ', 0];
fname  = coder.nullcopy(uint8(zeros(1, 32)));
coder.ceval('sprintf', coder.wref(fname), ['img%d.dat', 0], counter);
fd = coder.opaque('FILE *');
fd = coder.ceval('fopen', fname, ['w', 0]);
for i = 1:320
    for j = 1:240
        coder.ceval('fprintf', fd, format, r(i, j));
    end
```

```
end
for i = 1:320
    for j = 1:240
        coder.ceval('fprintf', fd, format, g(i, j));
    end
end
for i = 1:320
    for j = 1:240
        coder.ceval('fprintf', fd, format, b(i, j));
    end
end
coder.ceval('fclose', fd);
```

The file can then be retrieved by MATLAB with the following commands

```
h = raspberrypi
h.connect
h.getFile(['/home', '/', h.UserName, '/img0.dat'])
img = load('img0.dat');
x = uint8(reshape(img,240,320,3));
image(x)
shg
```



For additional features of using the raspberrypi object to issue linux commands and functions please study the documentation.

P4.3 Optional: Calling C Function Libraries from Simulink

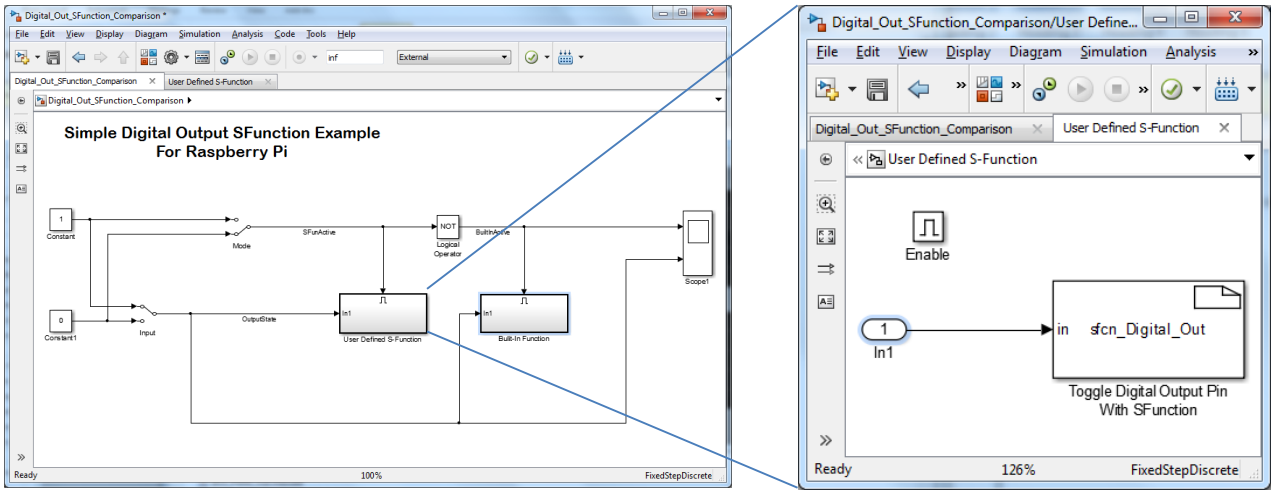
Objective: To demonstrate how to include calls to user written C functions and libraries

Task/ Challenge: To run and examine a Simulink model turning an LED on or off using both the block supplied in Simulink Library and calling functions from wiring Pi C library supplied with Raspberry Pi.

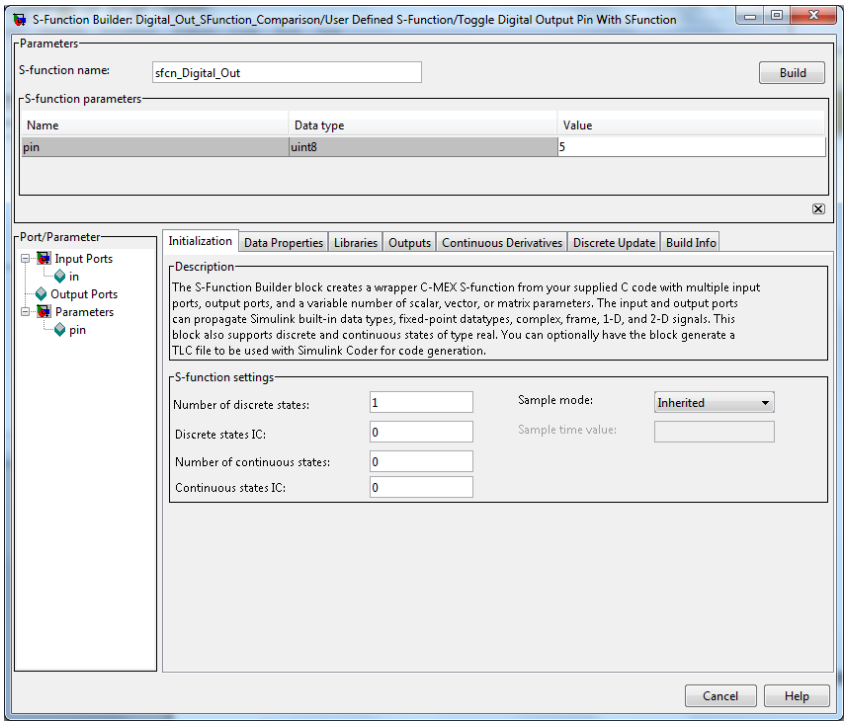
Steps/ Approach:

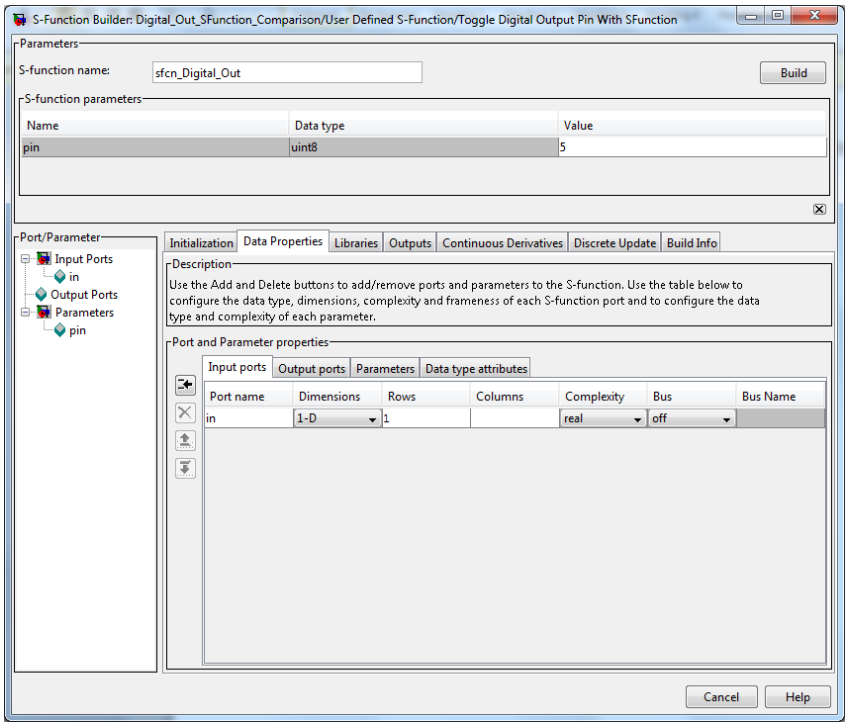
1. It is also possible to include call "C" functions which drive various outputs. In the following example we will use the Simulink S Function Builder alongside the wiring Pi "C" library which allows a different way of addressing the IO using user-defined libraries and drivers.
2. Details of how to include "C" functions with hardware support for Simulink can be found in the a document entitled "Device Drivers" http://www.mathworks.co.uk/matlabcentral/fileexchange/39354-device-drivers written for support for arduino but is equally applicable for Raspberry Pi.
3. In order to check how this is done copy the folder **"home"** and file **"Digital_Out_SFunction_Comparison.slx"** from the "Models\Appendix4" directory to the "Working" directory. The **home** folder contains the "C" include and source files as defined in wiringPi library.
4. Open **Digital_Out_SFunction_Comparison.slx**. Double click on User Defined S-Function. Double click on sfcn_Digital_Out to open up the dialogue box. Click on the Build button on the top right hand side. This

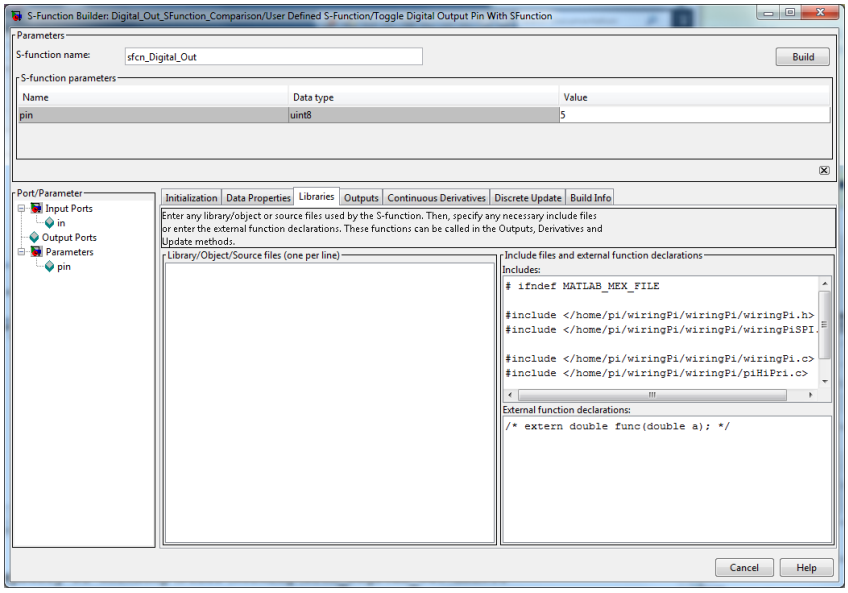
will create the appropriate files which will be compiled and called from Simulink to turn the Digital Output on using calls to user-defined “C” functions.

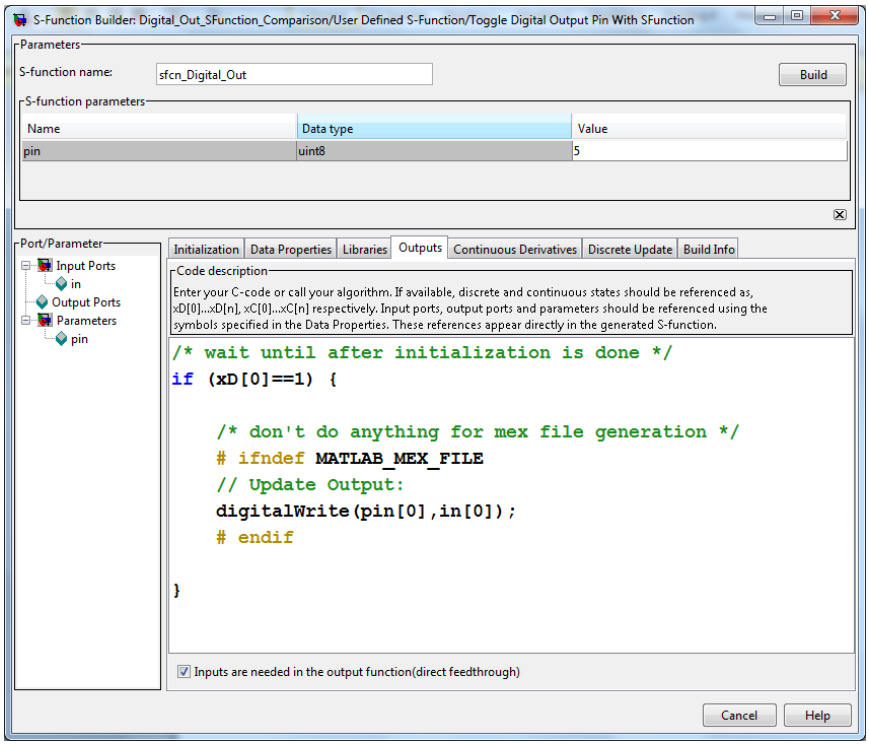


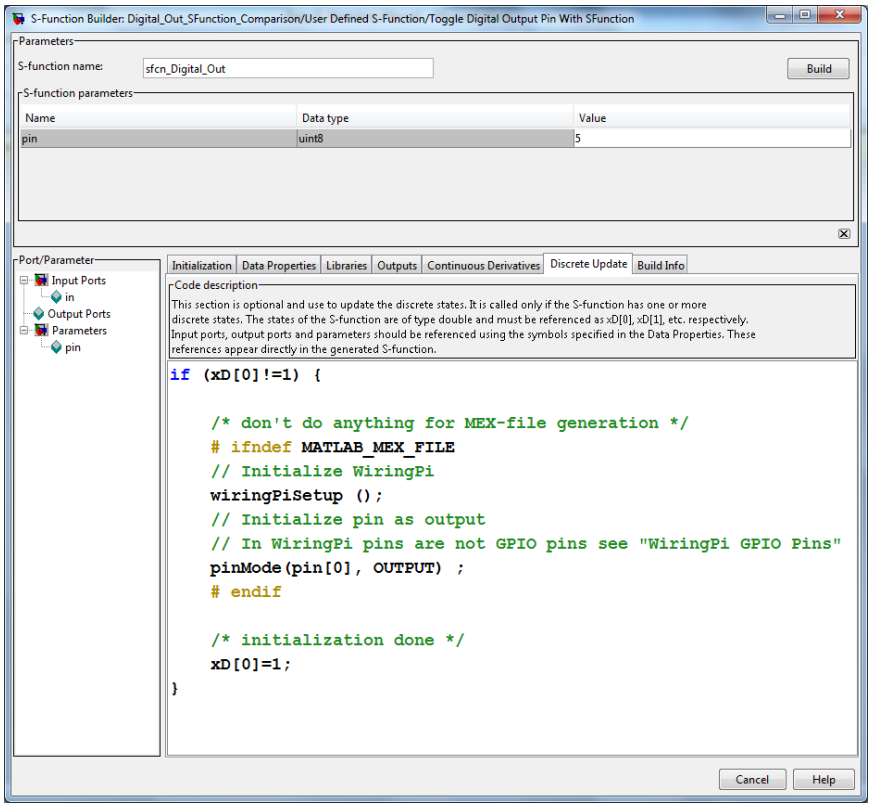
5. The Block with S-Function Builder is given by:




S-Function Builder: Digital_Out_SFunction_Comparison/User Defined S-Function/Toggle Digital Output Pin With SFunction
Parameters
S-function name:
sfcn_Digital_Out
Build
S-function parameters
Name
Data type
Value
pin
uint8
5
Port/Parameter
Input Ports
in
Output Ports
Parameters
pin
Initialization
Data Properties
Libraries
Outputs
Continuous Derivatives
Discrete Update
Build Info
Description
Use the Add and Delete buttons to add/remove ports and parameters to the S-function. Use the table below to configure the data type, dimensions, complexity and frameness of each S-function port and to configure the data type and complexity of each parameter.
Port and Parameter properties
Input ports
Output ports
Parameters
Data type attributes
Port name
Dimensions
Rows
Columns
Complexity
Bus
Bus Name
in
1-D
1
real
off
Cancel
Help


S-Function Builder: Digital_Out_SFunction_Comparison/User Defined S-Function/Toggle Digital Output Pin With SFunction
Parameters
S-function name:
sfcn_Digital_Out
Build
S-function parameters
Name
Data type
Value
pin
uint8
5
Port/Parameter
Input Ports
in
Output Ports
Parameters
pin
Initialization
Data Properties
Libraries
Outputs
Continuous Derivatives
Discrete Update
Build Info
Enter any library/object or source files used by the S-function. Then, specify any necessary include files or enter the external function declarations. These functions can be called in the Outputs, Derivatives and Update methods.
Library/Object/Source files (one per line)
Include files and external function declarations
Includes:
ifndef MATLAB_MEX_FILE
#include </home/pi/wiringPi/wiringPi/wiringPi.h>
#include </home/pi/wiringPi/wiringPi/wiringPiSPI.
#include </home/pi/wiringPi/wiringPi/wiringPi.c>
#include </home/pi/wiringPi/wiringPi/piHiPri.c>
External function declarations:
/* extern double func(double a); */
Cancel
Help


S-Function Builder: Digital_Out_SFunction_Comparison/User Defined S-Function/Toggle Digital Output Pin With SFunction
Parameters
S-function name:
sfcn_Digital_Out
Build
S-function parameters
Name
Data type
Value
pin
uint8
5
Port/Parameter
Input Ports
in
Output Ports
Parameters
pin
Initialization
Data Properties
Libraries
Outputs
Continuous Derivatives
Discrete Update
Build Info
Code description
Enter your C-code or call your algorithm. If available, discrete and continuous states should be referenced as, xD[0]...xD[n], xC[0]...xC[n] respectively. Input ports, output ports and parameters should be referenced using the symbols specified in the Data Properties. These references appear directly in the generated S-function.
/* wait until after initialization is done */
if (xD[0]==1) {
/* don't do anything for mex file generation */
ifndef MATLAB_MEX_FILE
// Update Output:
digitalWrite(pin[0],in[0]);
# endif
}
Inputs are needed in the output function(direct feedthrough)
Cancel
Help


S-Function Builder: Digital_Out_SFunction_Comparison/User Defined S-Function/Toggle Digital Output Pin With SFunction
Parameters
S-function name:
sfcn_Digital_Out
Build
S-function parameters
Name
Data type
Value
pin
uint8
5
Port/Parameter
Input Ports
in
Output Ports
Parameters
pin
Initialization
Data Properties
Libraries
Outputs
Continuous Derivatives
Discrete Update
Build Info
Code description
This section is optional and use to update the discrete states. It is called only if the S-function has one or more discrete states. The states of the S-function are of type double and must be referenced as xD[0], xD[1], etc. respectively. Input ports, output ports and parameters should be referenced using the symbols specified in the Data Properties. These references appear directly in the generated S-function.
if (xD[0]!=1) {
/* don't do anything for MEX-file generation */
ifndef MATLAB_MEX_FILE
// Initialize WiringPi
wiringPiSetup ();
// Initialize pin as output
// In WiringPi pins are not GPIO pins see "WiringPi GPIO Pins"
pinMode(pin[0], OUTPUT) ;
endif
/* initialization done */
xD[0]=1;
}
Cancel
Help

6. Examine the various tabs of the S Function Builder and note how the C functions are called. Build the S Function and run the application in External Mode. Note the way you can switch and use this block in building your system.

This demonstrates a second way which external C functions can be incorporated into Simulink. WiringPi functions can therefore be used simply to integrate new IO and drivers with Raspberry Pi blocks in Simulink.