Assignment A-5

1 Aim

Build a small compute cluster using Raspberry Pi/BBB modules to implement Booths Multiplication algorithm.

2 Objective

- Understand the working behind Booth's Multiplication Algorithm.
- Understand how socket programming works.

3 Software and Hardware Requirements

- *nix OS (64-bit)
- Beagle Bone Black (BBB)
- Python 2.7.11
- Connecting wires

4 Mathematical Model

4.1 Representations

Let S be the solution perspective of the given problem statement, where S is:
S = {s, e, X, Y, F_{me}, F, DD, NDD, S_C, F_C}

s is the start state.
s = {ip, port}
ip = [0-255].[0-255].[0-255].[0-255]

port = [1024-66536]

e is the end state.
e = {e_1,e_2}
e_1 = output of multiplication is obtained.
e_2 = connections are removed.

X are the set of inputs.
X = { x_1, x_2}
x_1 = first input number.
x_2 = second input number.

Y are the set of outputs.
Y = { y_1}
y_1 = product of the two input numbers.

F_{me} is the main function.
F are the set of functions used in the program.
F = { $F_{inverse}$,F_{add},$F_{display}$}
$F_{inverse}$ = Finds the one's complement of the number.
Example: 0110 $\rightarrow$ 1001
F_{add} = Adds the binary numbers and ignoring the last carry.
$F_{display}$ = Displays the result after multiplying.

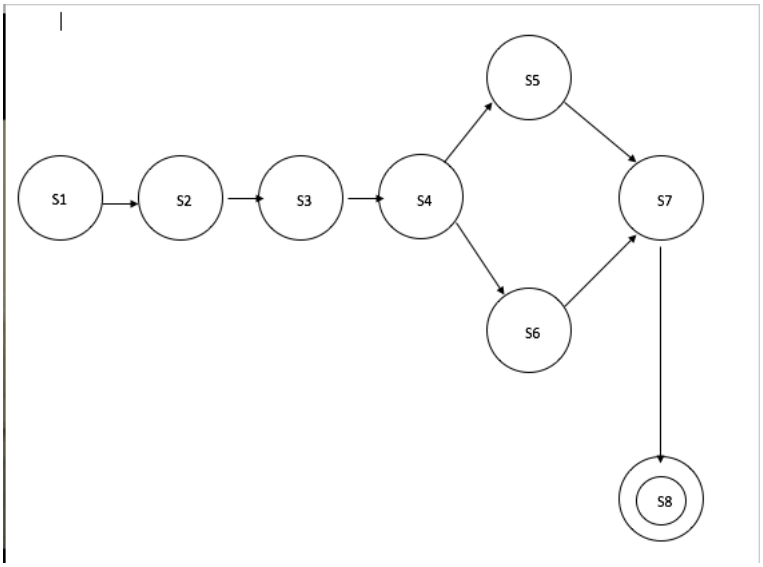
DD is the Deterministic Data.
DD = { Product when two numbers in a specific range are multiplied. }

NDD is the Non Deterministic Data.
NDD = {Product when the number cannot be represented in 64-bit format
}

S_C are the success cases.
S_C = { S_1, S_2 }
S_1 $\rightarrow$ 2 integers are multiplied within a range.
S_1 $\rightarrow$ Numbers are fetched from BBB when it is connected successfully.

F_C are the failure cases.
F_C = { F_1, F_2 }
F_1 $\rightarrow$ Numbers multiplied when out of range show wrong results.
F_1 $\rightarrow$ BBB not connected properly.

4.2 State Diagram



S1 → Start State
S2 → Connect State
S3 → Input Number State
S4 → Calculate Multiplier Matrix State
S5 → PC Compute State
S6 → BBB Compute State
S7 → Combine and Add State
S8 → Display/End State

5 Theory

5.1 Booth's Multiplication

Overview Booth's multiplication algorithm is a multiplication algorithm that multiplies two signed binary numbers in two's complement notation. The algorithm was invented by Andrew Donald Booth in 1950 while doing research on crystallography at Birkbeck College in Bloomsbury, London. Booth used desk calculators that were faster at shifting than adding and created the algorithm to increase their speed. Booth's algorithm is of interest in the study of computer architecture.

Algorithm Booth's algorithm examines adjacent pairs of bits of the N-bit multiplier Y in signed two's complement representation, including an implicit bit below the least significant bit, $y_{-1} = 0$. For each bit y_i, for i running from 0 to N-1, the bits y_i and y_{i-1} are considered. Where these two bits are equal, the product accumulator P is left unchanged. Where $y_i = 0$ and $y_{i-1} = 1$, the multiplicand times 2i is added to P; and where $y_i = 1$ and $y_{i-1} = 0$, the multiplicand times 2i is subtracted from P. The final value of P is the signed product.

The representations of the multiplicand and product are not specified; typically, these are both also in two's complement representation, like the multiplier, but any number system that supports addition and subtraction will work as well. As stated here, the order of the steps is not determined. Typically, it proceeds from LSB to MSB, starting at i = 0; the multiplication by 2i is then typically replaced by incremental shifting of the P accumulator to the right between steps; low bits can be shifted out, and subsequent additions and subtractions can then be done just on the highest N bits of P. There are many variations and optimizations on these details.

The algorithm is often described as converting strings of 1s in the multiplier to a high-order +1 and a low-order 1 at the ends of the string. When a string runs through the MSB, there is no high-order +1, and the net effect is interpretation as a negative of the appropriate value.

5.2 Beaglebone Black

Overview The BeagleBoard is a low-power open-source hardware single-board computer produced by Texas Instruments in association with Digi-Key and Newark element14. The BeagleBoard was also designed with open source software development in mind, and as a way of demonstrating the Texas Instrument's OMAP3530 system-on-a-chip.

Beaglebone the BeagleBone is a barebone development board with a Sitara ARM Cortex-A8 processor running at 720 MHz, 256 MB of RAM, two 46-pin expansion connectors, on-chip Ethernet, a microSD slot, and a USB host port and multipurpose device port which includes low-level serial control and JTAG hardware debug connections, so no JTAG emulator is required.

Hardware Specifications

- AM335x 1GHz ARM Cortex-A8
- 512MB DDR3 RAM

- 4GB 8-bit eMMC on-board flash storage
- 3D graphics accelerator
- NEON floating-point accelerator
- 2x PRU 32-bit microcontrollers

Software Compatibility

- Debian
- Android
- Ubuntu
- Cloud9

Connectivity

- USB client for power and communications
- USB host
- Ethernet
- HDMI
- 2x 46 pinheaders

6 Conclusion

Thus we have built a small compute cluster using BBB to implement Booths Multiplication algorithm.