microProcessor Dice Game

By: Matthew Guirguis and Chris Mathewson

Description

This dice game incorporates fighting game mechanics like hitting and dealing damage. The goal is to roll a higher attack roll than the enemy armor class. Once that is true, the ability to deal damage is available.

The microprocessor is able to handle multiple instructions calculating rolls using different character stats.

On the side chance that a perfect 20 out of 20 is rolled, the attack roll is instantly won, and double damage is dealt.

Stats

STRENGTH - how strong you are
DEXTERITY - how quick you are
INTELLIGENCE - how smart you are
WISDOM - how aware you are
CHARISMA - how attractive you are

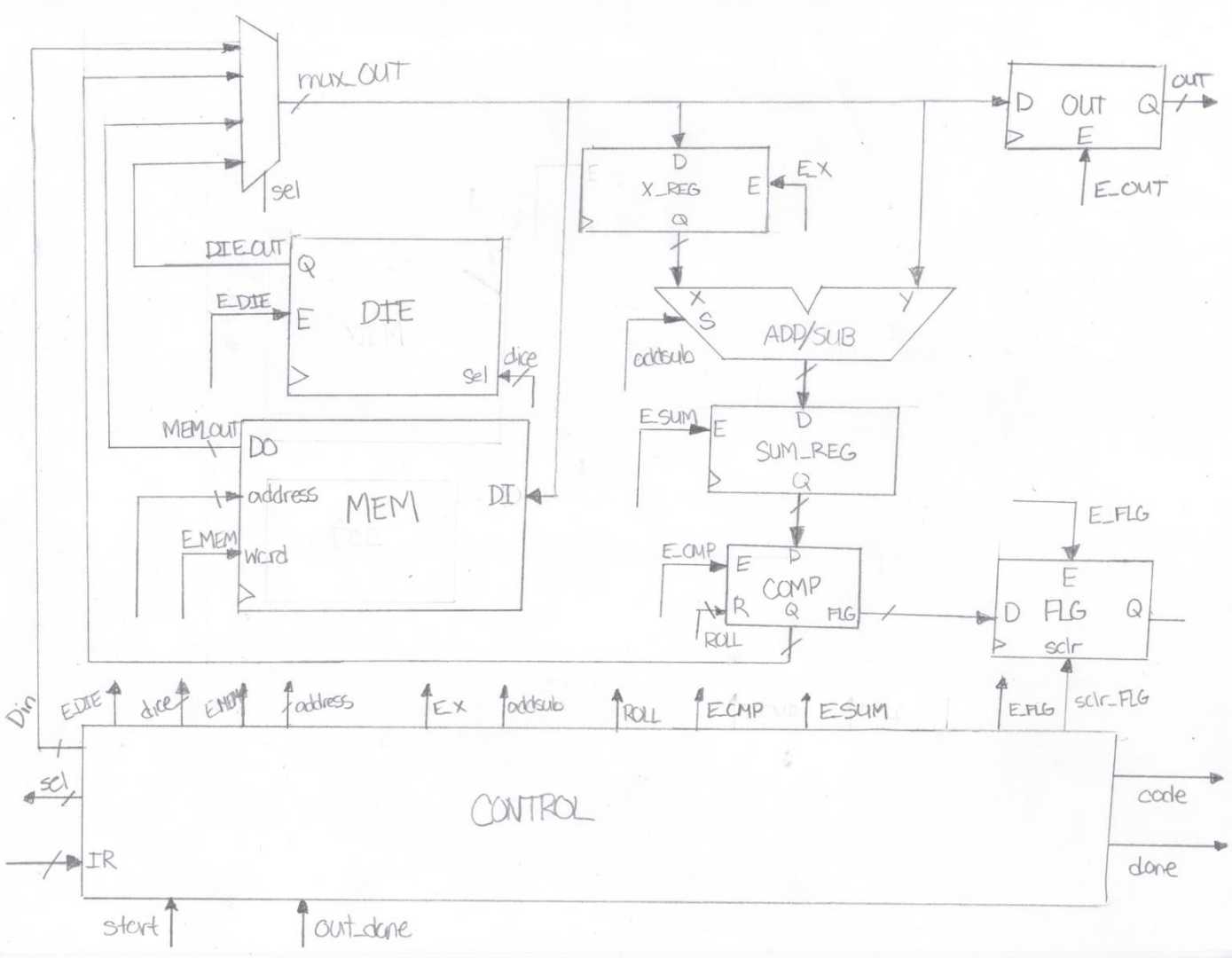
PROFICIENCY - your level bonus

ENEMY ARMOR CLASS - how hard your enemy is to hit

microProcessor

The microProcessor includes a memory ram emulator, nbit adder, an x register, an rsum register, an rout register, a flag register, a dice roller, a comparator and uP control file. There is also a mux with selector for the outputs

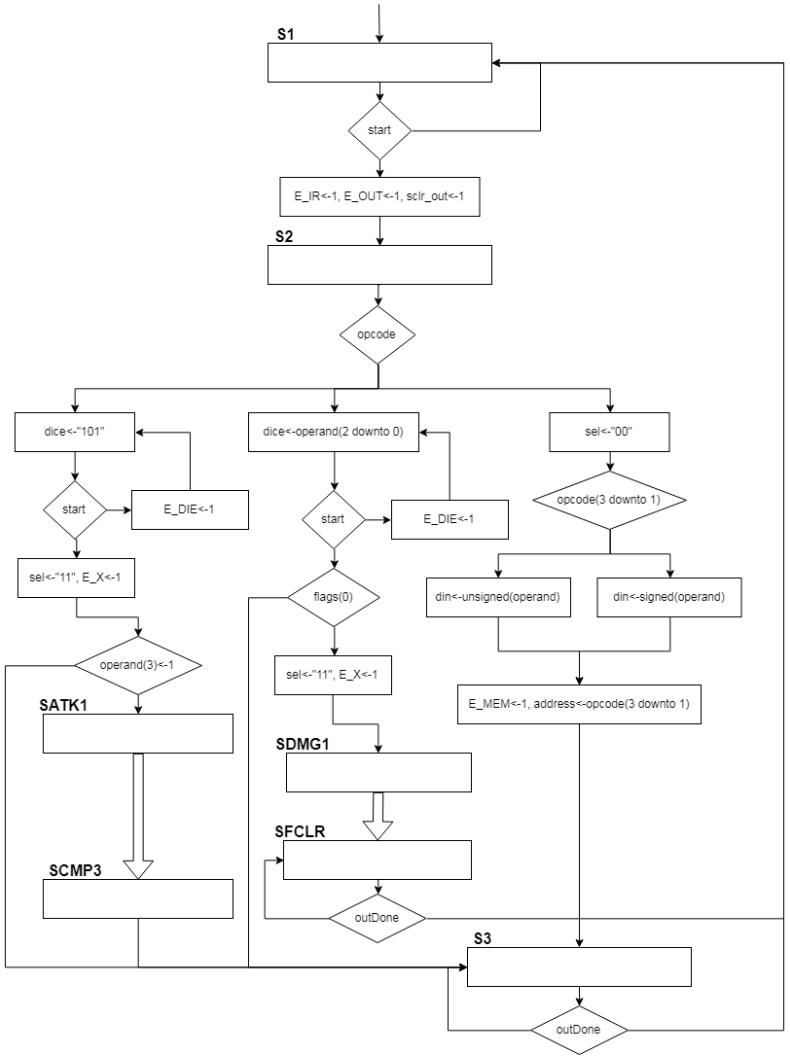
uProcessor


mux_OUT
sel
D
OUT
Q
E
OUT
E_OUT
D
X_REG
E
EX
Q
DIE_OUT
Q
E_DIE
E
DIE
sel
dice
X
S
Y
ADD/SUB
addsub
E_SUM
E
D
SUM_REG
Q
MEM_OUT
DO
address
MEM
DI
E_MEM
word
E_CMP
E
P
COMP
R
Q
FLG
ROLL
E_FLG
E
D
FLG
Q
sclr
Din
E_DIE
dice
E_MEM
address
EX
addsub
ROLL
E_CMP
E_SUM
E_FLG
sclr_FLG
scl
CONTROL
code
IR
done
start
out_done

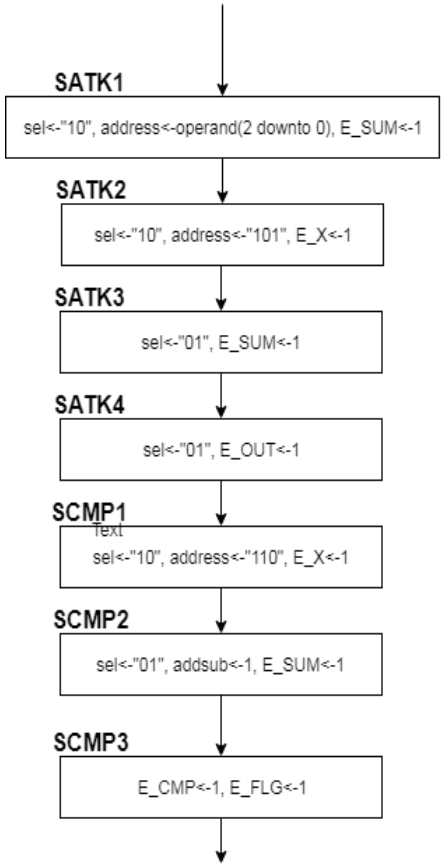
Control uP

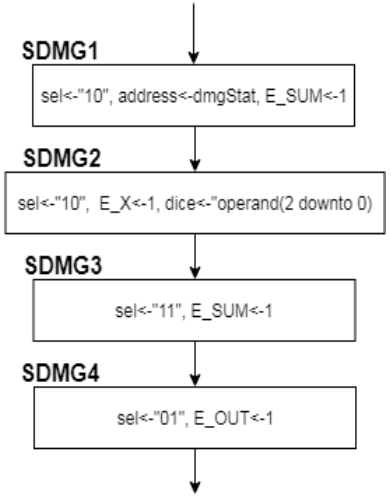
The control uP, is a finite state machine. There are 3 main states, 4 atk states for the roll, 3 compare states to determine how much health the enemy has, and 4 damage states to determine whether the hit was a critical hit.

uProcessor FSM



Instruction FSM

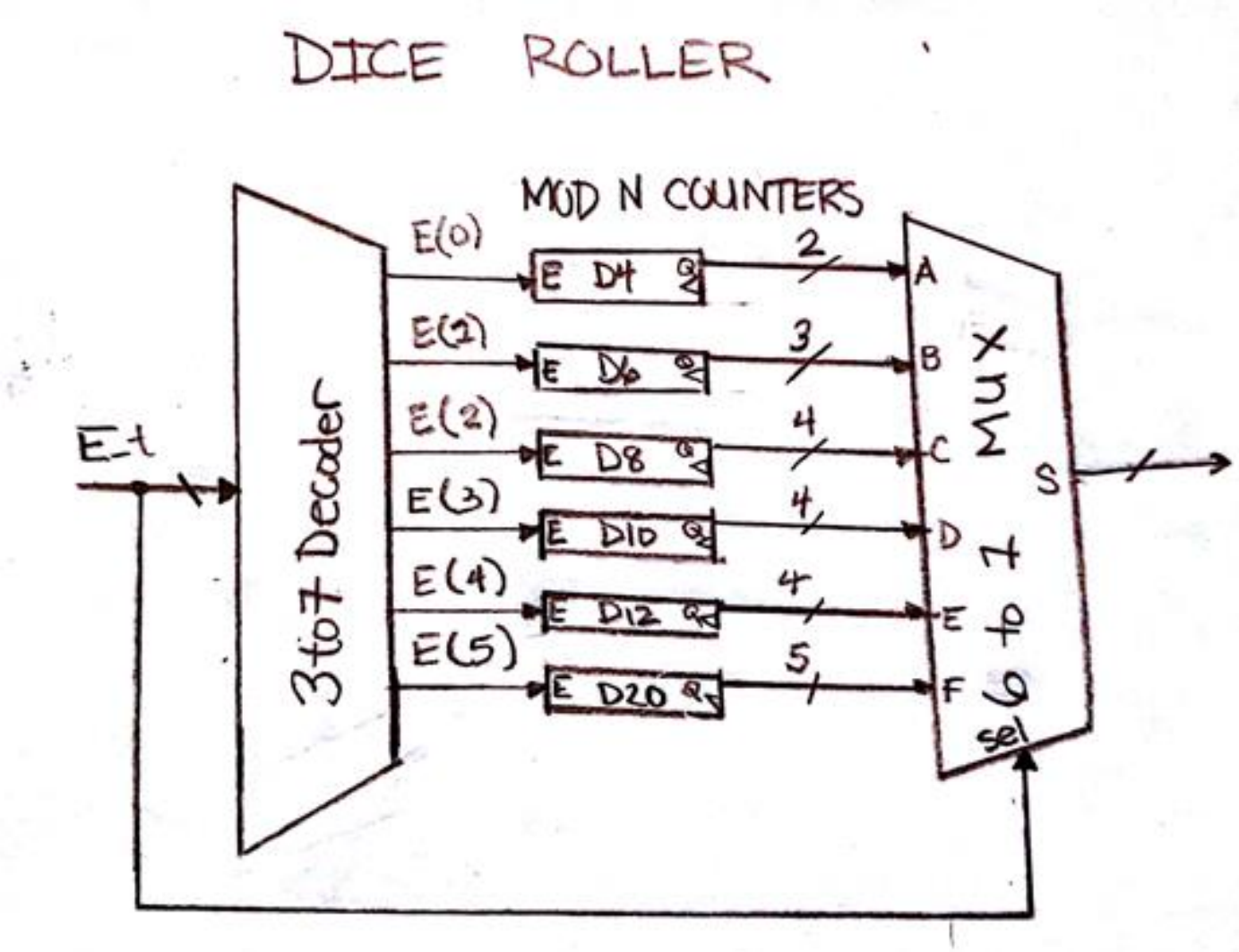




Dice Roller

The dice roller uses 6 modulo counters on a decoded enable. The output of each is sent to a multiplexer and selected by using the same enable code. The result is a set of a four, six, eight, ten, twelve, and twenty-sided dice for individual use.

Dice Roller

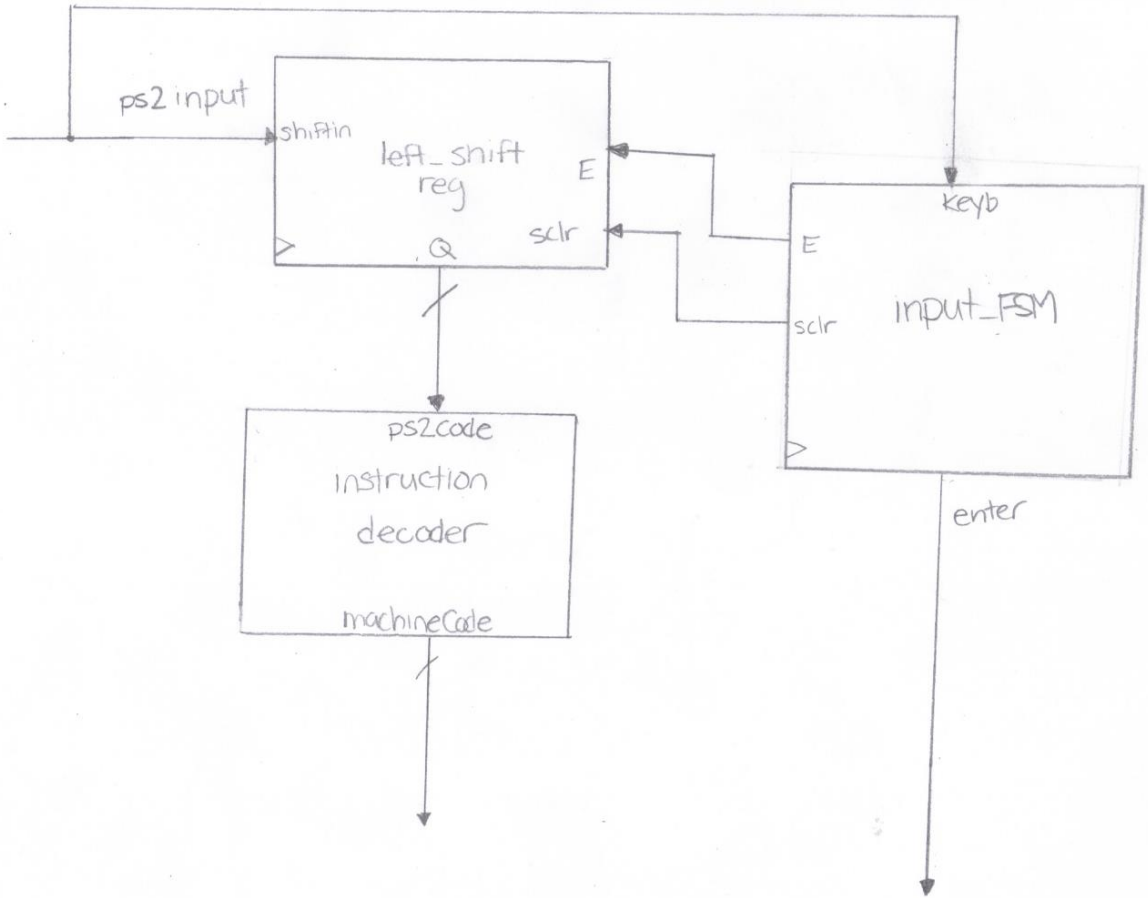

DICE ROLLER
MOD N COUNTERS
E(0)
E(1)
E(2)
E(3)
E(4)
E(5)
E D4 Q
E D6 Q
E D8 Q
E D10 Q
E D12 Q
E D20 Q
2
3
4
4
4
5
E-1
3to7 Decoder
A
B
C
D
E
F
sel
MUX 7 to 1
S

Assembler

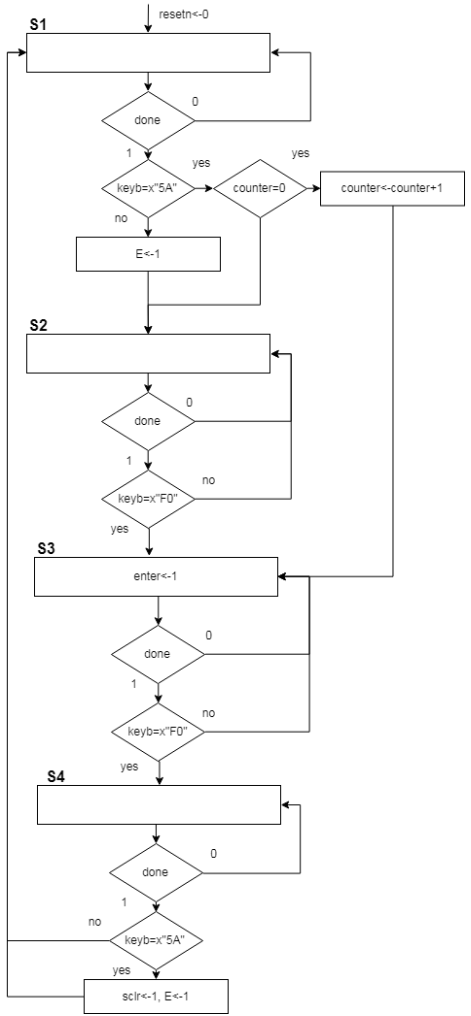
The Assembler includes a left shift register, instruction decoder and input finite state machine.

The purpose of the assembler is to convert the ps/2 codes received from the keyboard into machine code for the microprocessor to understand.

Assembler


ps2 input
shiftin
left_shift reg
E
Q
sclr
keyb
E
sclr
input_FSM
ps2code
instruction decoder
machineCode
enter

Assembler FSM



Dice Instruction List

Instruction	Machine Code
ROLLS	1110_0000
ROLLD	1110_0001
ROLLI	1110_0010
ROLLW	1110_0011
ROLLC	1110_0100

Damage Instruction List

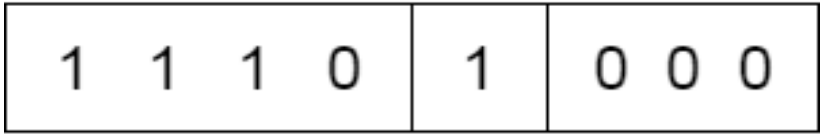
Instruction	Machine Code
DMGDG	1111_1000
DMGSB	1111_1001
DMGLS	1111_0010
DMGGS	1111_0100
DMGAX	1111_0010
DMGGA	1111_0100
DMGWH	1111_0100
DMGCB	1111_1011
DMGLB	1111_1010

Store Instruction List

Instruction	Machine Code
STR *op*	000 *op*
DEX *op*	001 *op*
INT *op*	010 *op*
WIS *op*	011 *op*
CHA *op*	100 *op*
PRF *op*	101 *op*
EAC *op*	110 *op*

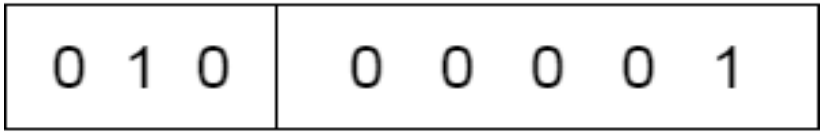
Instruction Format

Roll Instruction Format



Damage Instruction Format

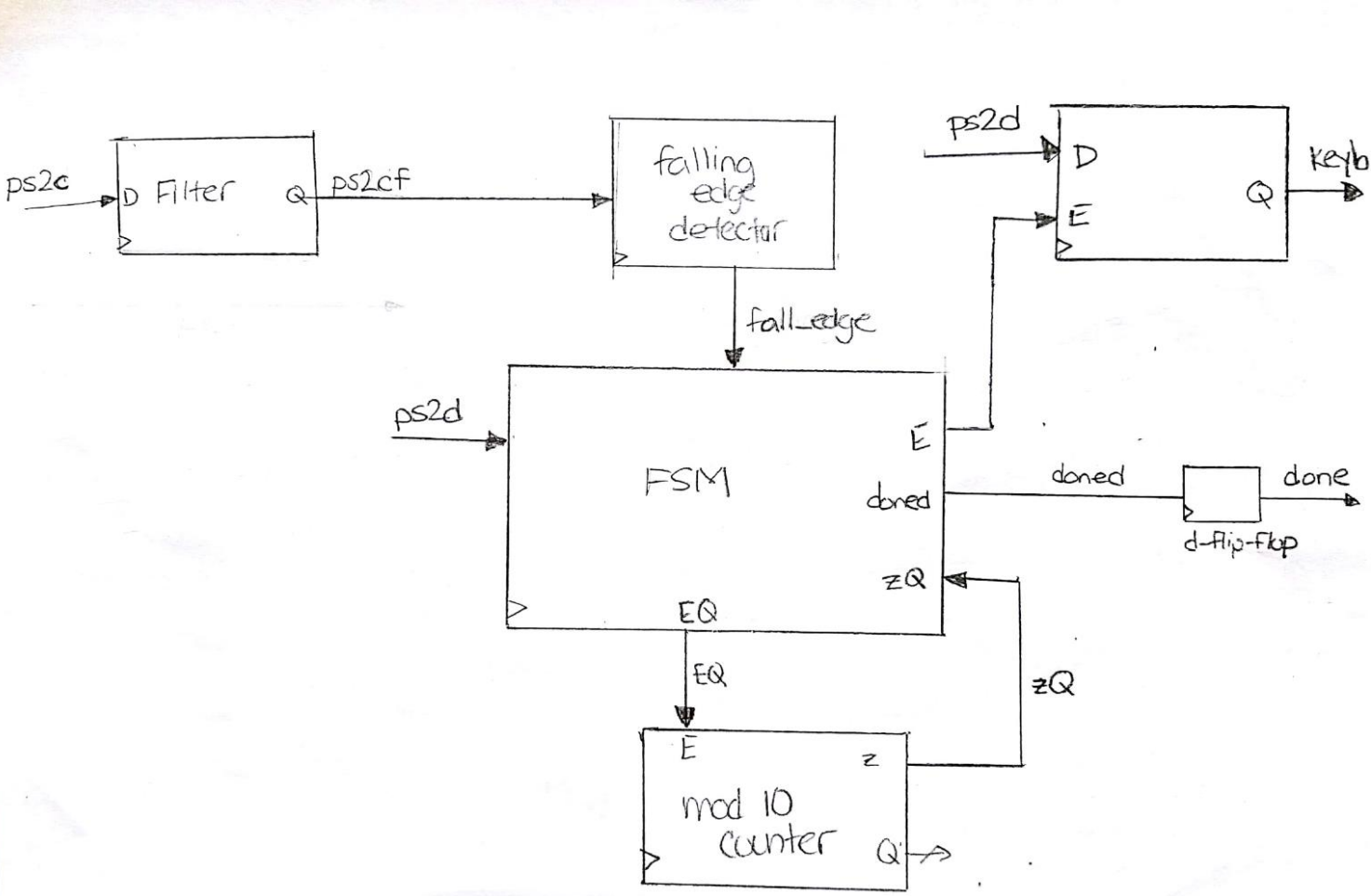
Stat Instruction Format

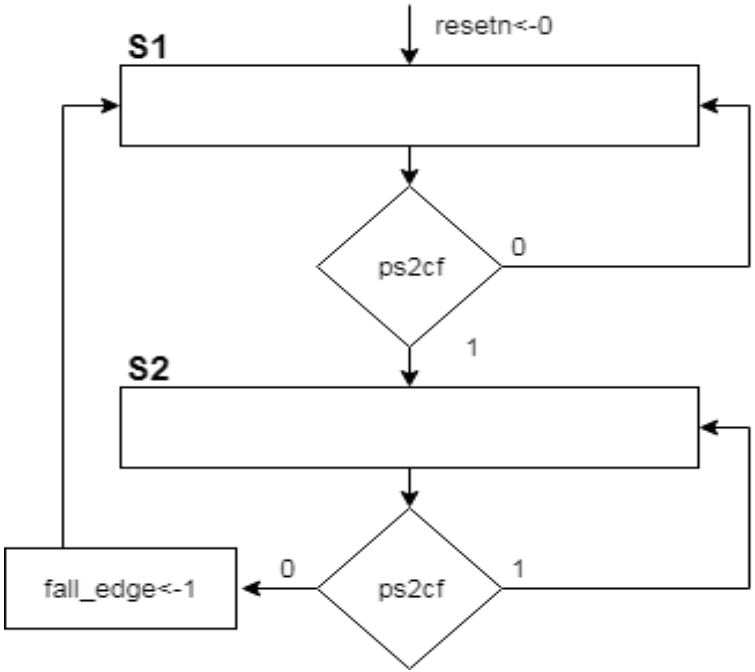


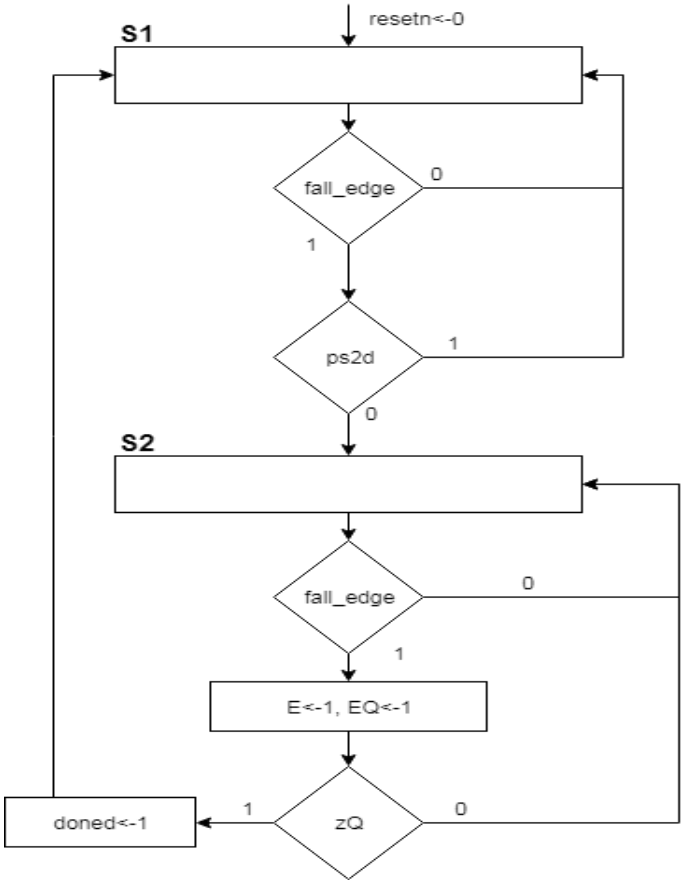
PS2keyboard

The ps2kyboard enables the use of a serial keyboard connected to the FPGA. The keyboard gets used to input the data by using PS/2. The PS2kyboard includes a genpulse file, parallel access shift register, ps2 filter, and d-flip-flop file.

PS2 keyboard







Falling Edge Detector

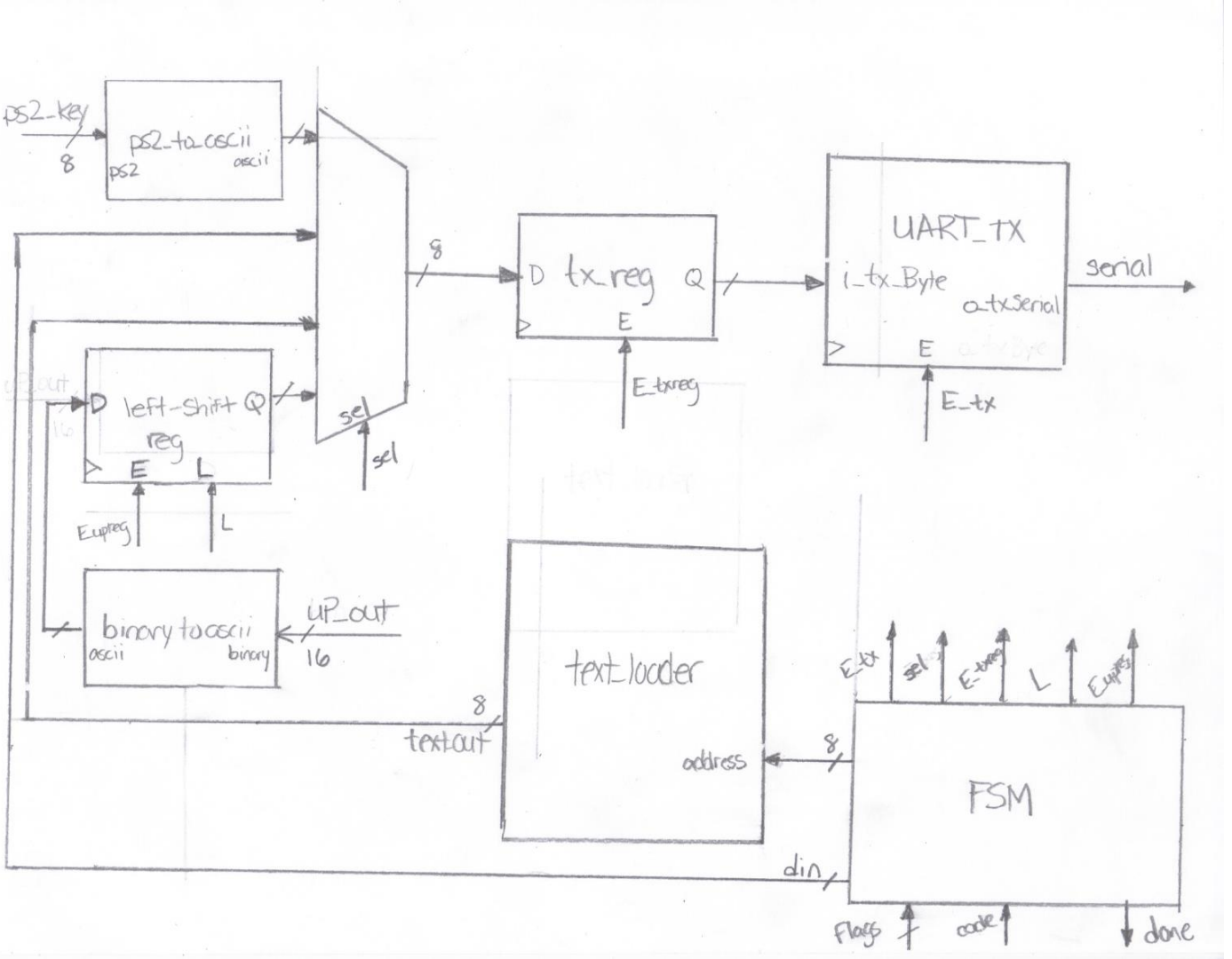
PS/2 Keyboard

Main FSM

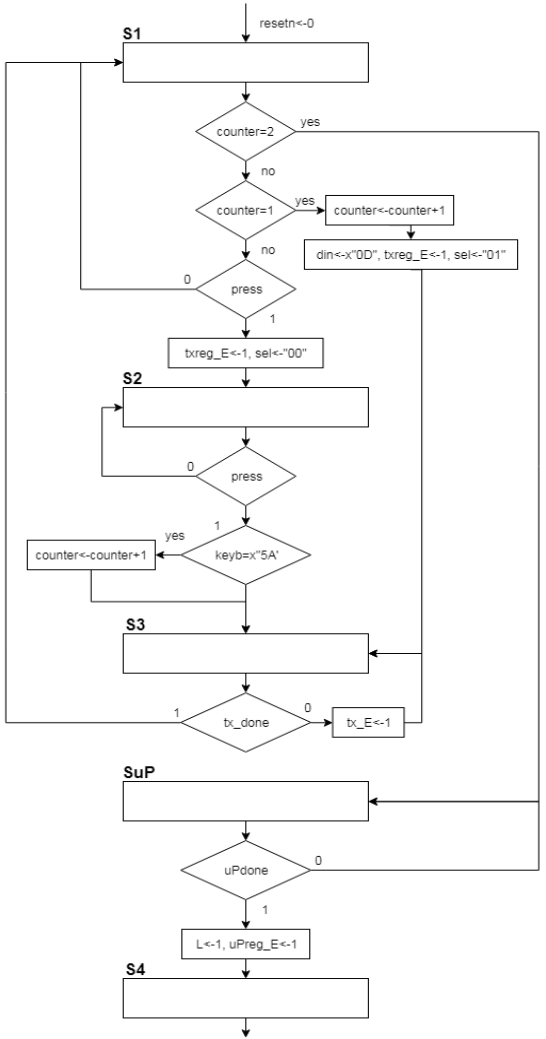
Output

The output file controls what is being outputted. This includes a text loader, UART transmitter, a transmitter register, uP left shift register, ps2 to ascii, and binary to ascii file.

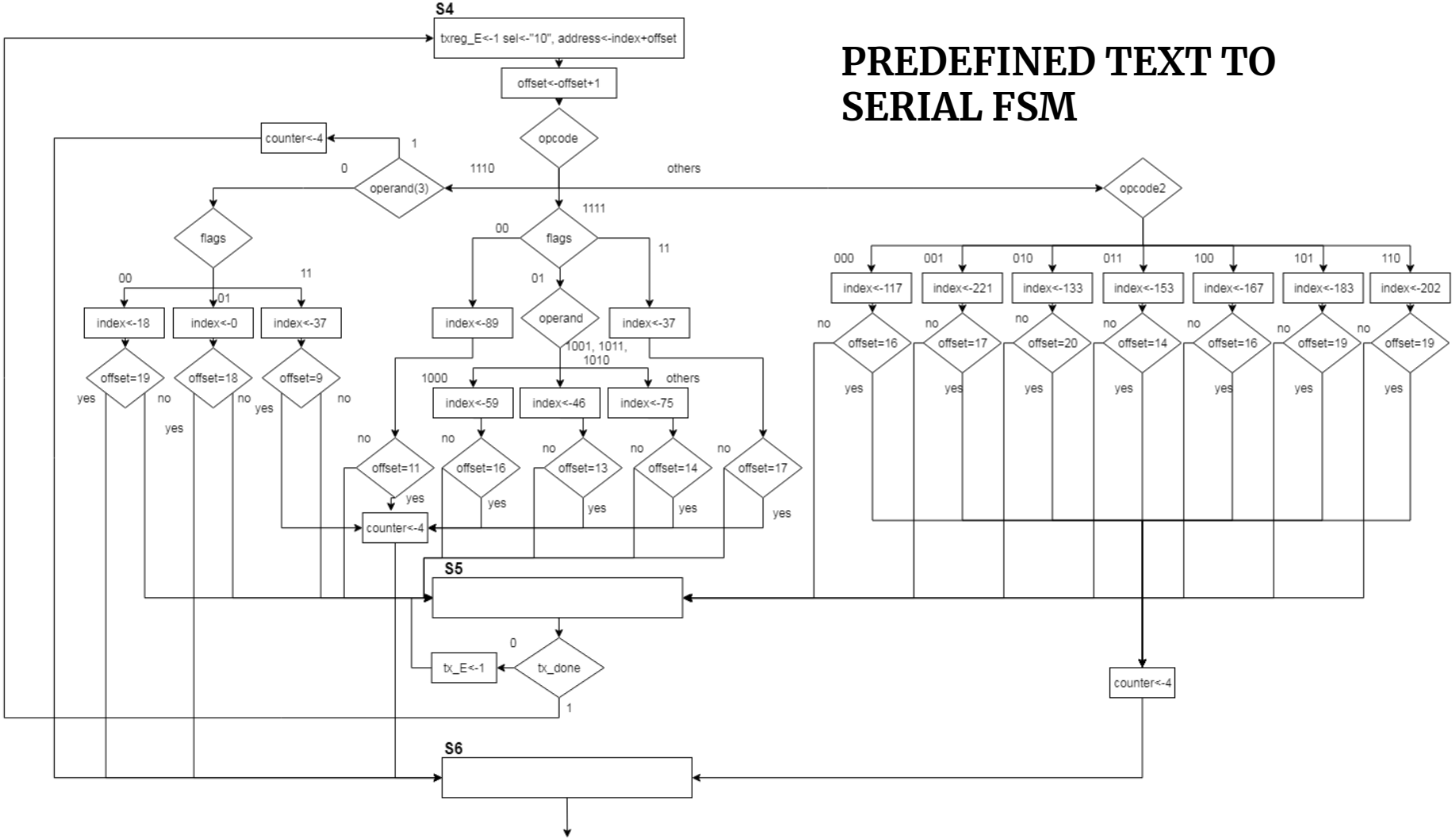
Output


ps2_key
8
ps2_to_ascii
ps2
ascii
8
D tx_reg Q
E
E_txreg
UART_TX
i_tx_Byte
o_txSerial
E
E_tx
serial
D left-shift Q
reg
E L
E_upreg
L
sel
sel
binary to ascii
ascii
binary
UP_out
16
text_loader
address
8
textout
8
FSM
din
Flags
code
done

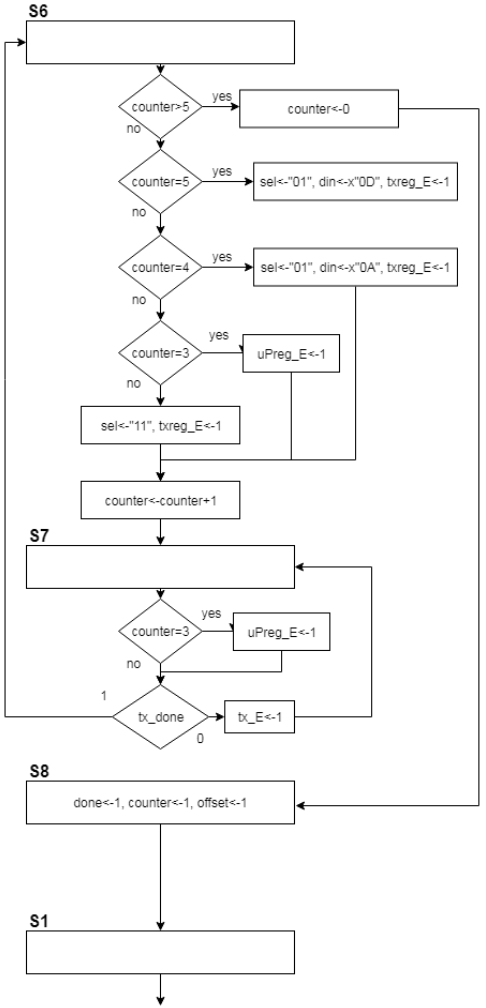
Output Keyboard To Serial FSM



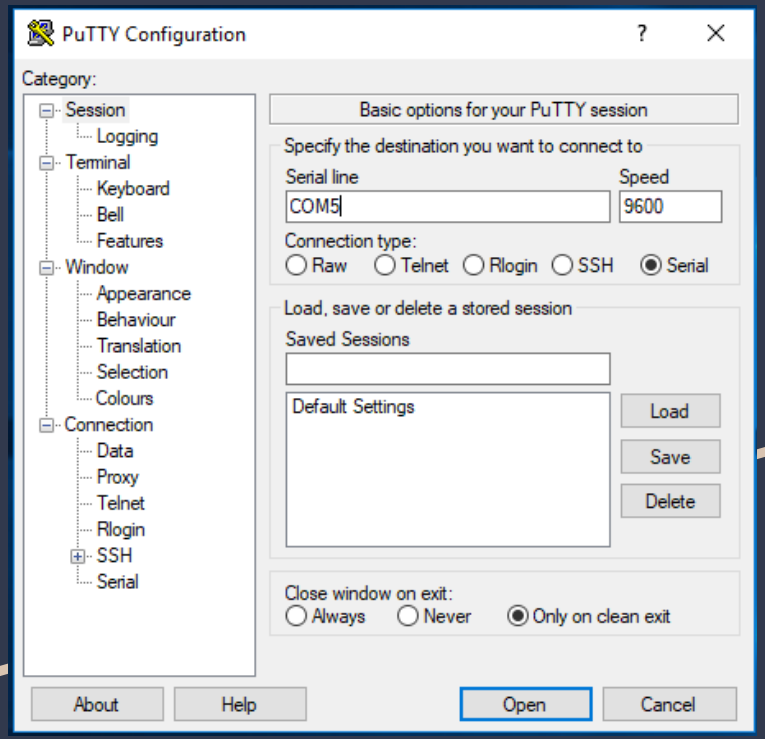
PREDEFINED TEXT TO SERIAL FSM



uProcessor Output To Serial FSM



UART_TX

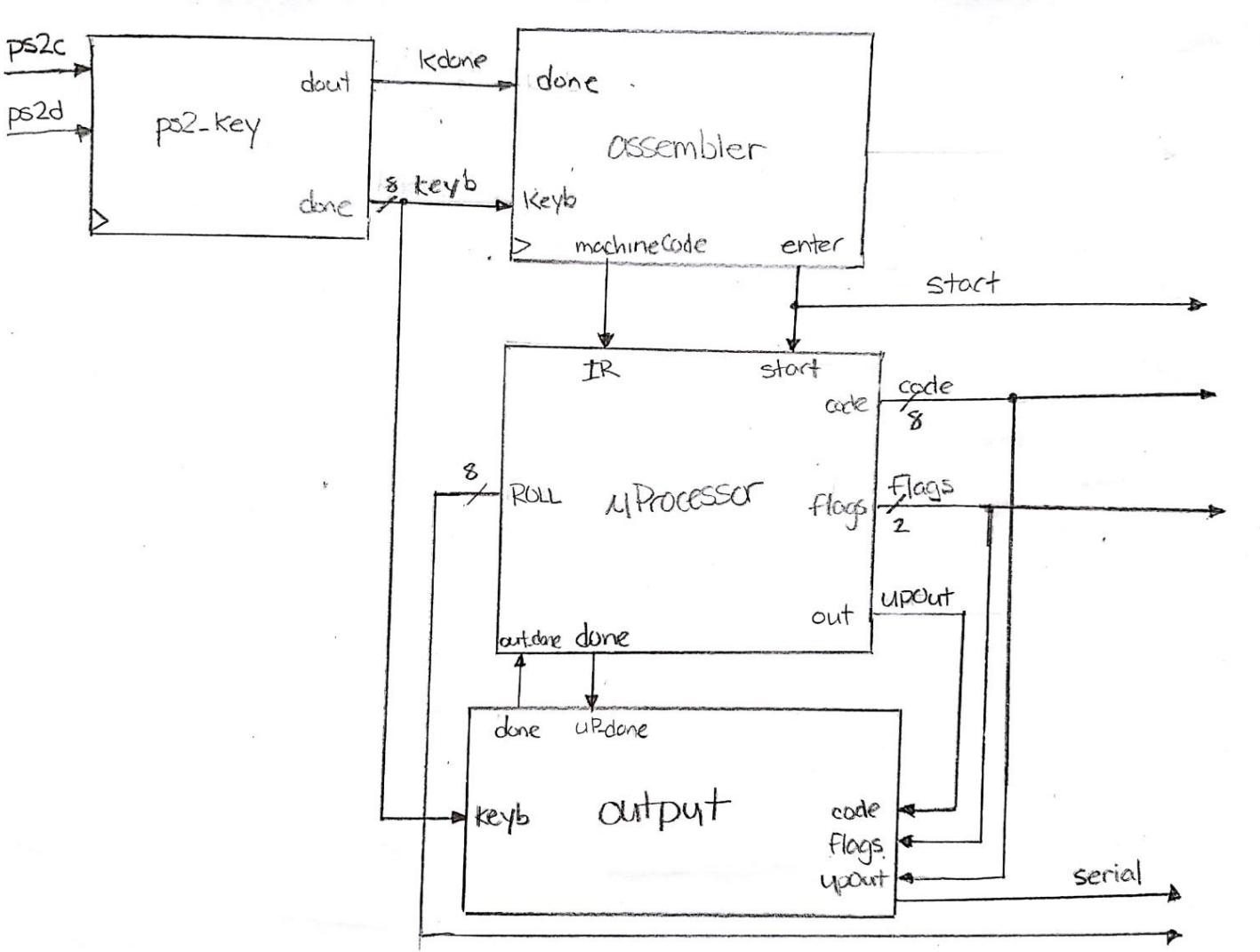


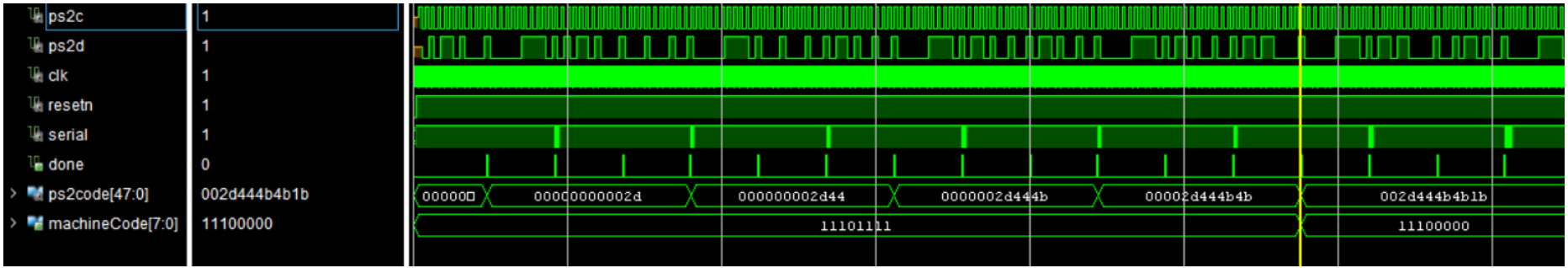
We used uart to transmit the signals to and from the board. We used a program called Putty as the source terminal for the UART protocol. This program asks for the speed and which COM port it is connected to. The baud rate is represented in the output control by setting the UART_TX fsm to 10416 clock ticks per read. After that no other information is needed to start up the source terminal.

UART_TX FSM

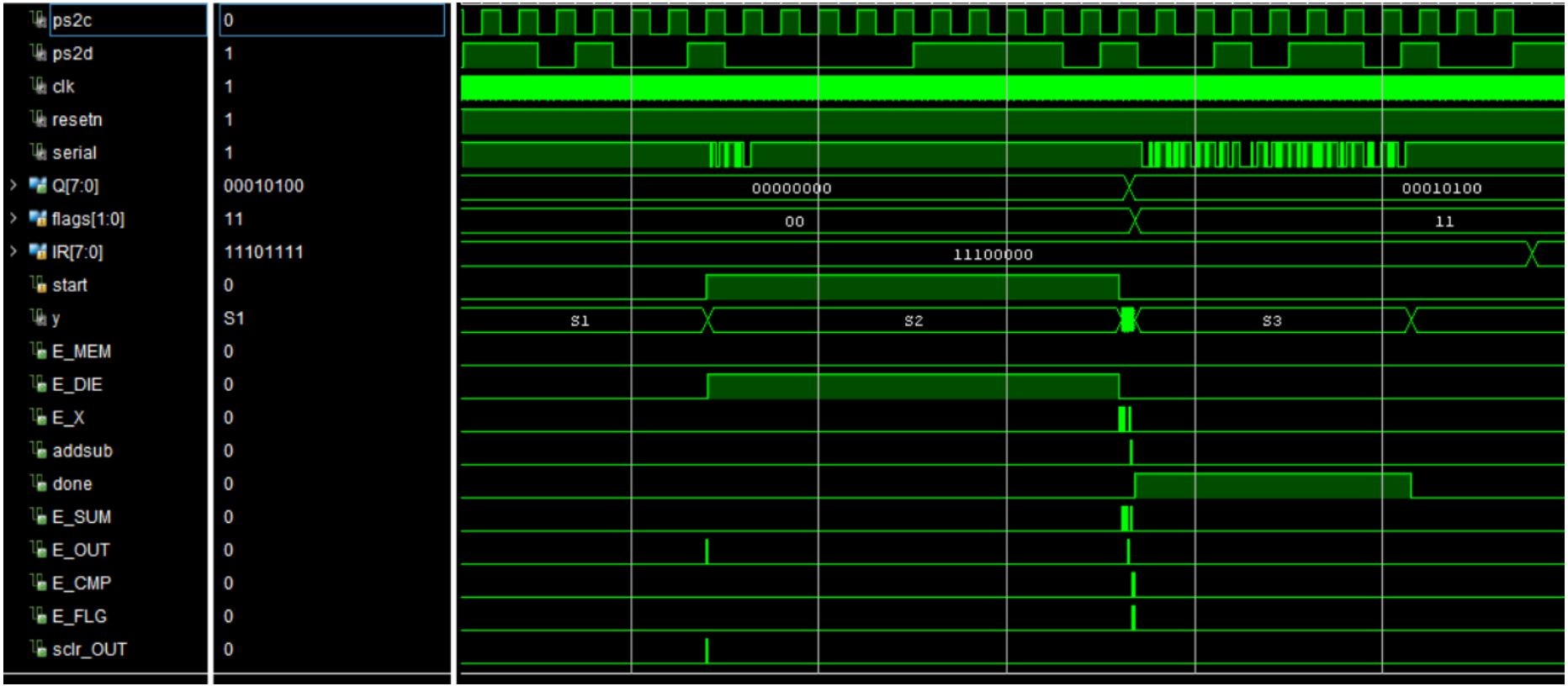

S_Idle
TX_Serial <= 1
i_TX_DV = 1
No
Yes
r_TX_Data <= i_TX_Byte
s_TX_Start_Bit
TX_Active <= 1
r_Clk_Count < g_Clks_Per_Bit - 1
Yes
r_Clk_Count <= r_Clk_Count + 1
No
r_Clk_Count <= 0
s_TX_Data_Bits
TX_Serial <= r_TX_Data(r_Bit_Index)
r_Clk_Count < g_Clks_Per_Bit - 1
Yes
r_Clk_Count <= r_Clk_Count + 1
No
r_Clk_Count <= 0
r_Bit_Index < 7
Yes
No
r_Bit_Index <= 0
s_Stop_Bit
TX_Serial <= 1
r_Clk_Count < g_Clks_Per_Bit - 1
Yes
r_Clk_Count <= r_Clk_Count + 1
No
r_TX_done <= 1
s_Cleanup
r_TX_Done <= 1

Top Level


ps2c
ps2d
ps2_key
dout
done
kdone
8 keyb
done
assembler
keyb
machineCode
enter
start
IR
start
code
code
8
8
ROLL
μProcessor
flags
flags
2
out
uPOut
outdone
done
done
uP_done
keyb
output
code
flags
upOut
serial


ps2c
1
ps2d
1
clk
1
resetn
1
serial
1
done
0
ps2code[47:0]
002d444b4b1b
machineCode[7:0]
11100000
00000□
000000000002d
000000002d44
0000002d444b
00002d444b4b
002d444b4b1b
11101111
11100000

TESTBENCH


ps2c
0
ps2d
1
clk
1
resetn
1
serial
1
Q[7:0]
00010100
00000000
00010100
flags[1:0]
11
00
11
IR[7:0]
11101111
11100000
start
0
y
S1
S1
S2
S3
E_MEM
0
E_DIE
0
E_X
0
addsub
0
done
0
E_SUM
0
E_OUT
0
E_CMP
0
E_FLG
0
sclr_OUT
0

TESTBENCH

Any Questions?