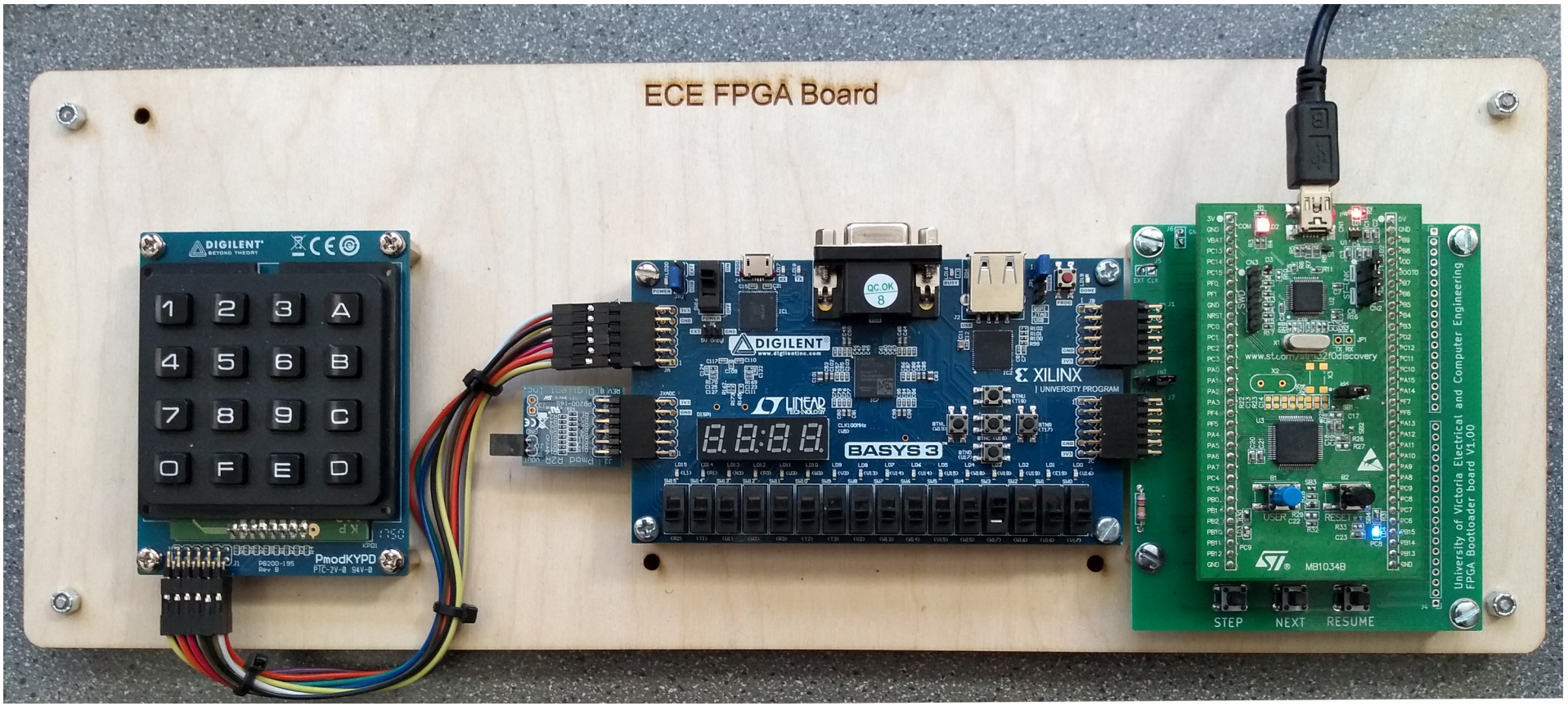
FPGA Boot Loader



Version 1.0 - March 20, 2019

Boot Loader Process

The purpose of the boot loader is to allow applications to be loaded remotely into your test CPU core's memory without having to rebuild the core. The process involves 5 steps.

- Assembling your test code
- Transferring the binary code to the STM32F0
- Resetting the test CPU in load mode (address 0x0002)
- Loading the code into your test CPU's memory
- Resetting the test CPU in execute mode (address 0x0000)

Assembling Your Code

To assemble you code you will be using the command line tool assembler19.py

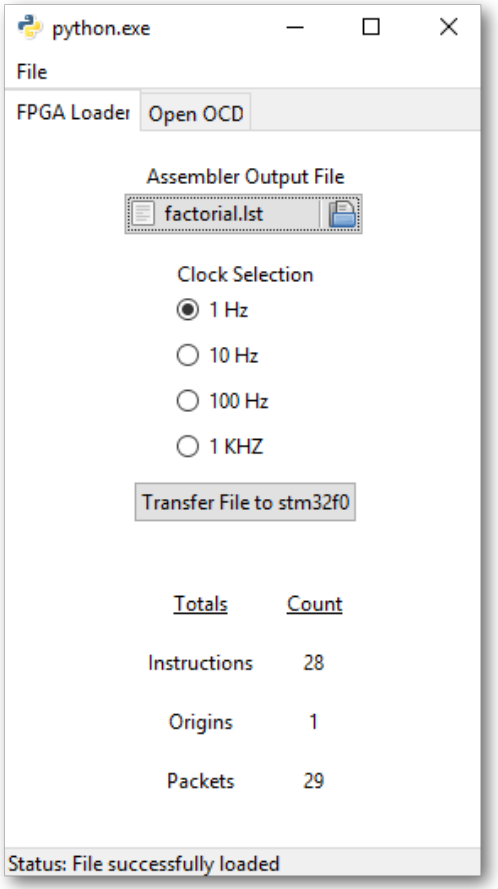
python assembler19.py -o factorial factorial.asm

- Leave the extension off the output file
- The assembler will create two files .LST and .HEX
- .LST is the standard listing file to be used with the FPGA loader python script
- .HEX is a file to import into ROM component (Boot loader code)

```
LedDisplay:     equ             0xFFF2
DipSwitches:    equ             0xFFF0
DipSwitchMask:  equ             31                      ; Binary multiple as a mask


                org             0x410
main:           loadimm.upper   DipSwitches.hi
                loadimm.lower   DipSwitches.lo

                load            r6, r7
                loadimm.upper   DipSwitchMask.hi
                loadimm.lower   DipSwitchMask.lo

                nand            r6, r6, r7              ; Nand the switch settings
                nand            r6, r6, r6              ; one's compliment result to and it


                loadimm.upper   0x00
                loadimm.lower   0x01
                mov             r4, r7
                mov             r3, r7

                test            r6
                brr.z           Done

                sub             r6, r6, r3              ; 0 and 1 factorial should return 1

                test            r6
                brr.z           Done

                loadimm.upper   0x00
                loadimm.lower   0x02
                mov             r5, r7

loop:           mul             r4, r4, r5
                add             r5, r5, r3

                sub             r6, r6, r3
                test            r6
                brr.z           Done
                brr             loop

Done:           loadimm.upper   LedDisplay.hi
                loadimm.lower   LedDisplay.lo

                store           r7, r4
                brr             Done

                end
```

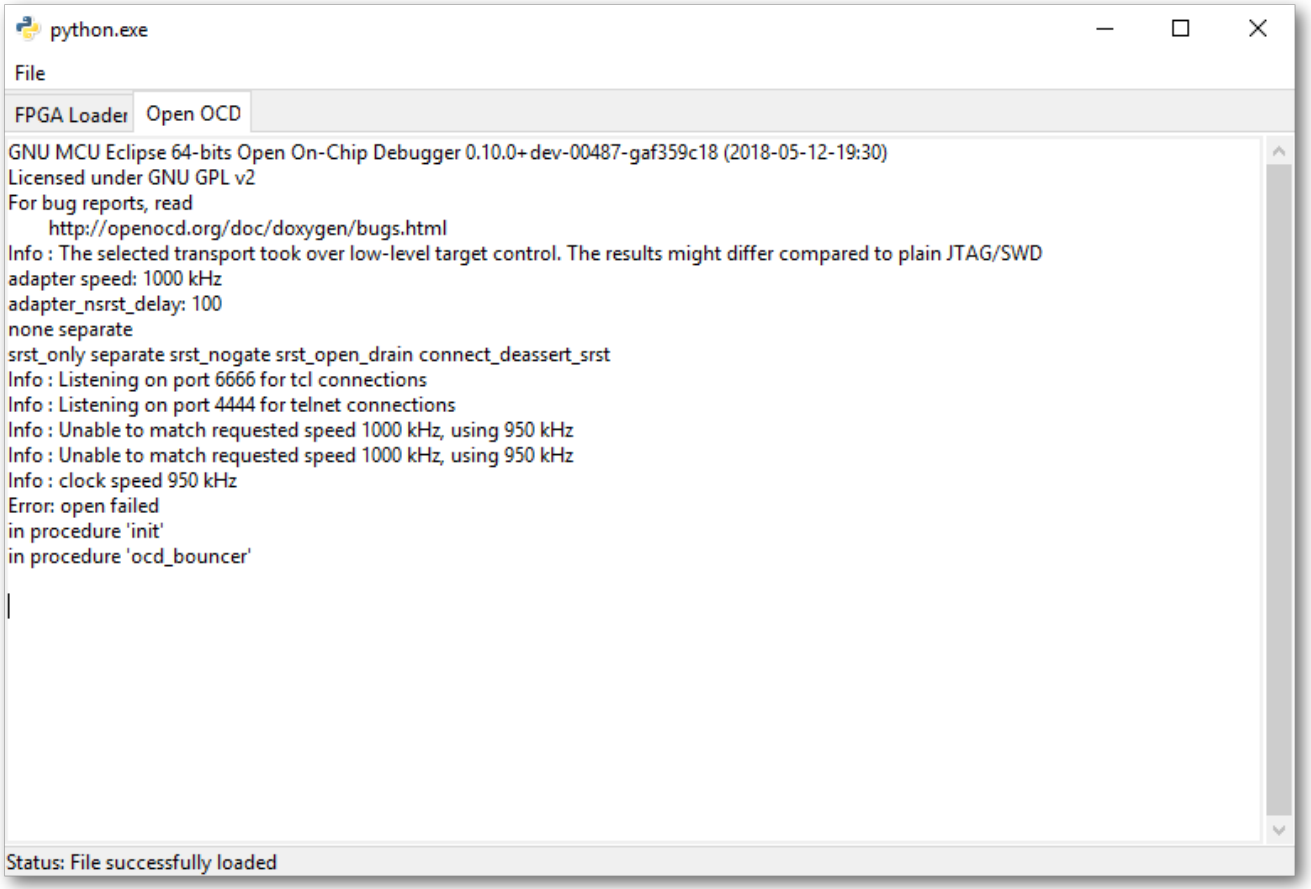
```
;
; Created on Wed Mar 20 13:31:56 2019 with ECE 449 assembler v1.10 Beta 4 (16 bit).
;
;Header Section
RADIX 10
DEPTH 2048
WIDTH 16
DEFAULT 0
;
; Data Section
; Specifies data to be stored in different addresses
; e.g., DATA 0:A, 1:0
;
RADIX 16
DATA
1040 => "0010010111111111", -- 0410 - 25FF main:          loadimm.upper DipSwitches.hi
1042 => "0010010011110000", -- 0412 - 24F0                loadimm.lower DipSwitches.lo
1044 => "0010000110111000", -- 0414 - 21B8                load          r6,r7
1046 => "0010010100000000", -- 0416 - 2500                loadimm.upper DipSwitchMask.hi
1048 => "0010010000011111", -- 0418 - 241F                loadimm.lower DipSwitchMask.lo
1050 => "0000100110110111", -- 041A - 09B7                nand          r6,r6,r7
1052 => "0000100110110110", -- 041C - 09B6                nand          r6,r6,r6
1054 => "0010010100000000", -- 041E - 2500                loadimm.upper 0x00
1056 => "0010010000000001", -- 0420 - 2401                loadimm.lower 0x01
1058 => "0010011100111000", -- 0422 - 2738                mov           r4,r7
1060 => "0010011011111000", -- 0424 - 26F8                mov           r3,r7
1062 => "0000111110000000", -- 0426 - 0F80                test          r6
1064 => "1000010000001101", -- 0428 - 840D                brr.z         Done
1066 => "0000010110110011", -- 042A - 05B3                sub           r6,r6,r3
1068 => "0000111110000000", -- 042C - 0F80                test          r6
1070 => "1000010000001010", -- 042E - 840A                brr.z         Done
1072 => "0010010100000000", -- 0430 - 2500                loadimm.upper 0x00
1074 => "0010010000000010", -- 0432 - 2402                loadimm.lower 0x02
1076 => "0010011101111000", -- 0434 - 2778                mov           r5,r7
1078 => "0000011100100101", -- 0436 - 0725 loop:          mul           r4,r4,r5
1080 => "0000011101101011", -- 0438 - 036B                add           r5,r5,r3
1082 => "0000010110110011", -- 043A - 05B3                sub           r6,r6,r3
1084 => "0000111110000000", -- 043C - 0F80                test          r6
1086 => "1000010000000010", -- 043E - 8402                brr.z         Done
1088 => "1000000111111011", -- 0440 - 81FB                brr           loop
1090 => "0010010111111111", -- 0442 - 25FF Done:          loadimm.upper LedDisplay.hi
1092 => "0010010011110010", -- 0444 - 24F2                loadimm.lower LedDisplay.lo
1094 => "0010001111100000", -- 0446 - 23E0                store         r7,r4
1096 => "1000000111111101", -- 0448 - 81FD                brr           Done


-------------------------------------------------------
Symbol Table:
DipSwitchMask      31 (001F)
DipSwitches     65520 (FFF0)
Done             1090 (0442)
LedDisplay      65522 (FFF2)
loop             1078 (0436)
main             1040 (0410)
```

Transferring Test Code with FPGA Loader

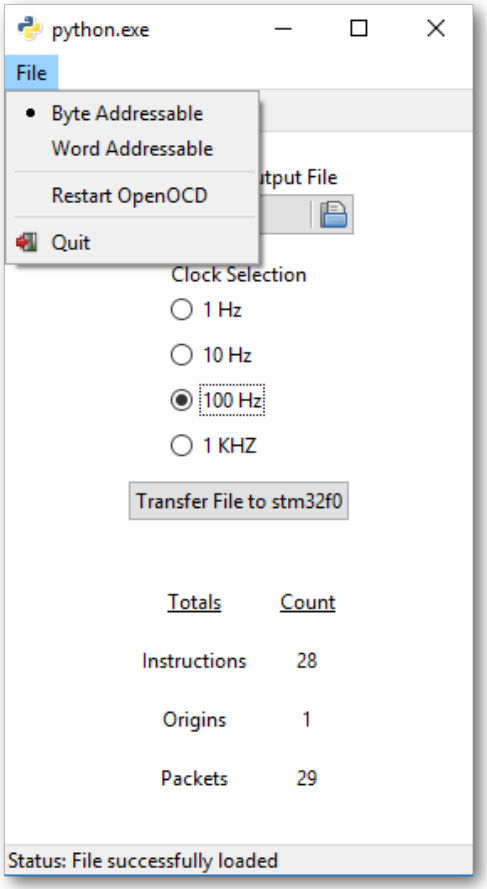


Location of link to program will be on your desktop in a ece449 folder.

Debugging Openocd



Menu Options

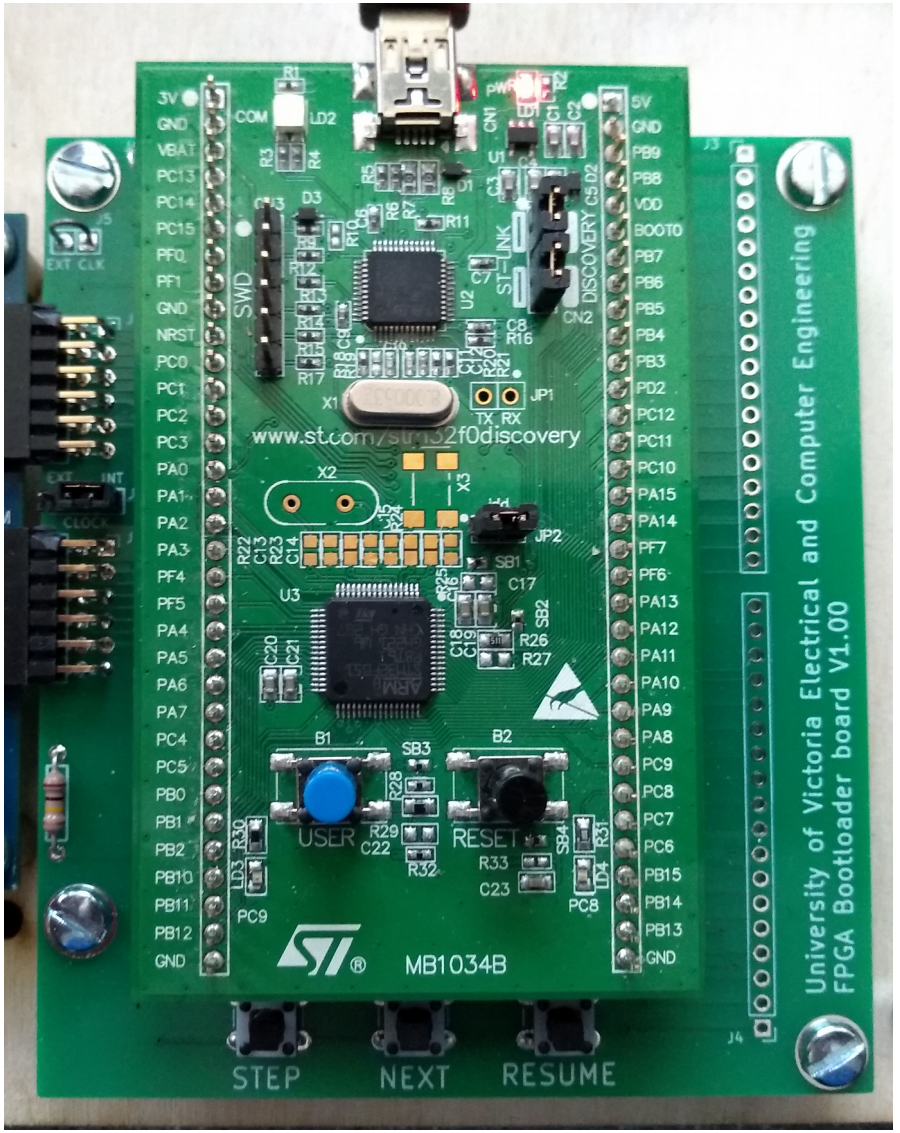


Packets

Each packet can contain either an address or data.

- Address packets are inserted at the start of the code and after each ORG statement
- Data packets are inserted for each instruction

STM32F0DISCOVERY Board



Board Features

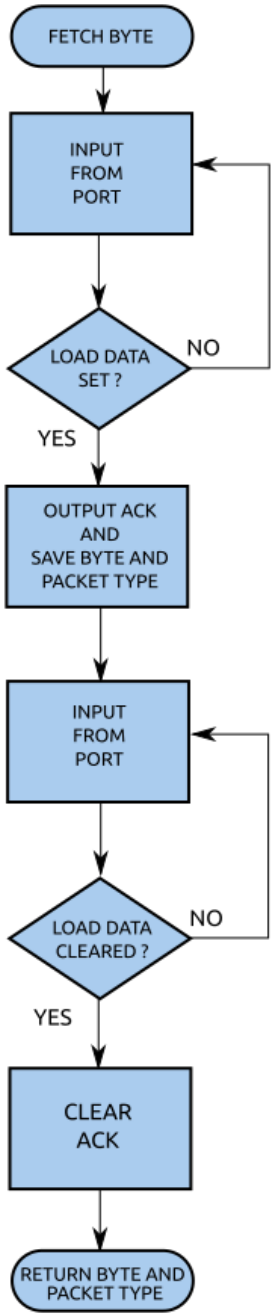
- Step - manually clock the FPGA board. Push down for rising edge and release for falling.
- Next - keep clock running at the selected frequency until ACK signal received
- Resume - continuously run the clock at the selected frequency. Pushing the button again will stop the clock
- STM32F0 Reset - Restart the FPGA loader application. The test code loaded will remain intact.
- External clock can be applied by attach signal to wire loops in upper left hand corner of the board. You will need to move the jumper from internal position to the external. The jump is located in the middle of the board on the left side.

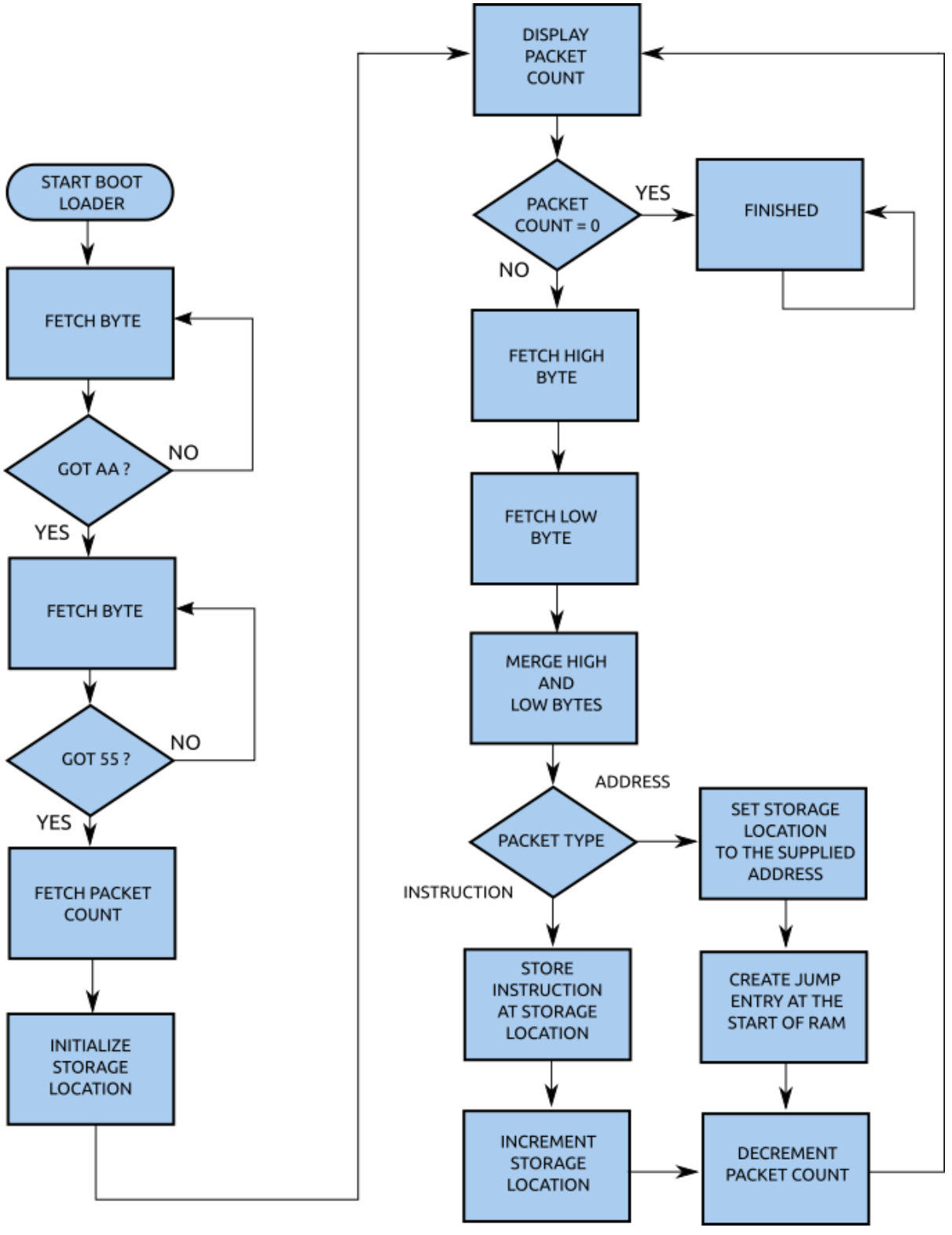
STM32F0 to FPGA Signals

Boot Loader Signals from STM32F0 Board				
STM32F0 SIGNALS	SIGNAL DIRECTION	LOGIC VECTOR	PMOD PIN	FPGA PIN
SYSTEM CLOCK	OUTPUT	STD LOGIC	JC1	K17
LOAD	OUTPUT	INPUT PORT(7)	JC8	R18
ADDRESS/DATA	OUTPUT	INPUT PORT(6)	JC7	P17
ACKNOWLEDGE	INPUT	OUTPUT PORT(0)	JC2	M18
DATA BIT 7	OUTPUT	INPUT PORT(15)	JB8	C16
DATA BIT 6	OUTPUT	INPUT PORT(14)	JB7	C15
DATA BIT 5	OUTPUT	INPUT PORT(13)	JB6	A17
DATA BIT 4	OUTPUT	INPUT PORT(12)	JB5	A15
DATA BIT 3	OUTPUT	INPUT PORT(11)	JB4	B16
DATA BIT 2	OUTPUT	INPUT PORT(10)	JB3	B15
DATA BIT 1	OUTPUT	INPUT PORT(9)	JB2	A16
DATA BIT 0	OUTPUT	INPUT PORT(8)	JB1	A14

Boot Loader Code

- Dynamically loads code into RAM memory
- Support for two reset vectors (reset load and reset execute)
- First three words in RAM memory are allocated to the boot loader.
- They contain three instruction to vector to the start of the test program. Boot loader validates these location when a reset execute occurs before vectoring to the code
- The last org in the code is the address that will contain the first instruction to be executed


FETCH BYTE
INPUT FROM PORT
LOAD DATA SET ?
NO
YES
OUTPUT ACK AND SAVE BYTE AND PACKET TYPE
INPUT FROM PORT
LOAD DATA CLEARED ?
NO
YES
CLEAR ACK
RETURN BYTE AND PACKET TYPE


START BOOT LOADER
FETCH BYTE
GOT AA ?
NO
YES
FETCH BYTE
GOT 55 ?
NO
YES
FETCH PACKET COUNT
INITIALIZE STORAGE LOCATION
DISPLAY PACKET COUNT
PACKET COUNT = 0
YES
FINISHED
NO
FETCH HIGH BYTE
FETCH LOW BYTE
MERGE HIGH AND LOW BYTES
ADDRESS
PACKET TYPE
SET STORAGE LOCATION TO THE SUPPLIED ADDRESS
INSTRUCTION
STORE INSTRUCTION AT STORAGE LOCATION
CREATE JUMP ENTRY AT THE START OF RAM
INCREMENT STORAGE LOCATION
DECREMENT PACKET COUNT