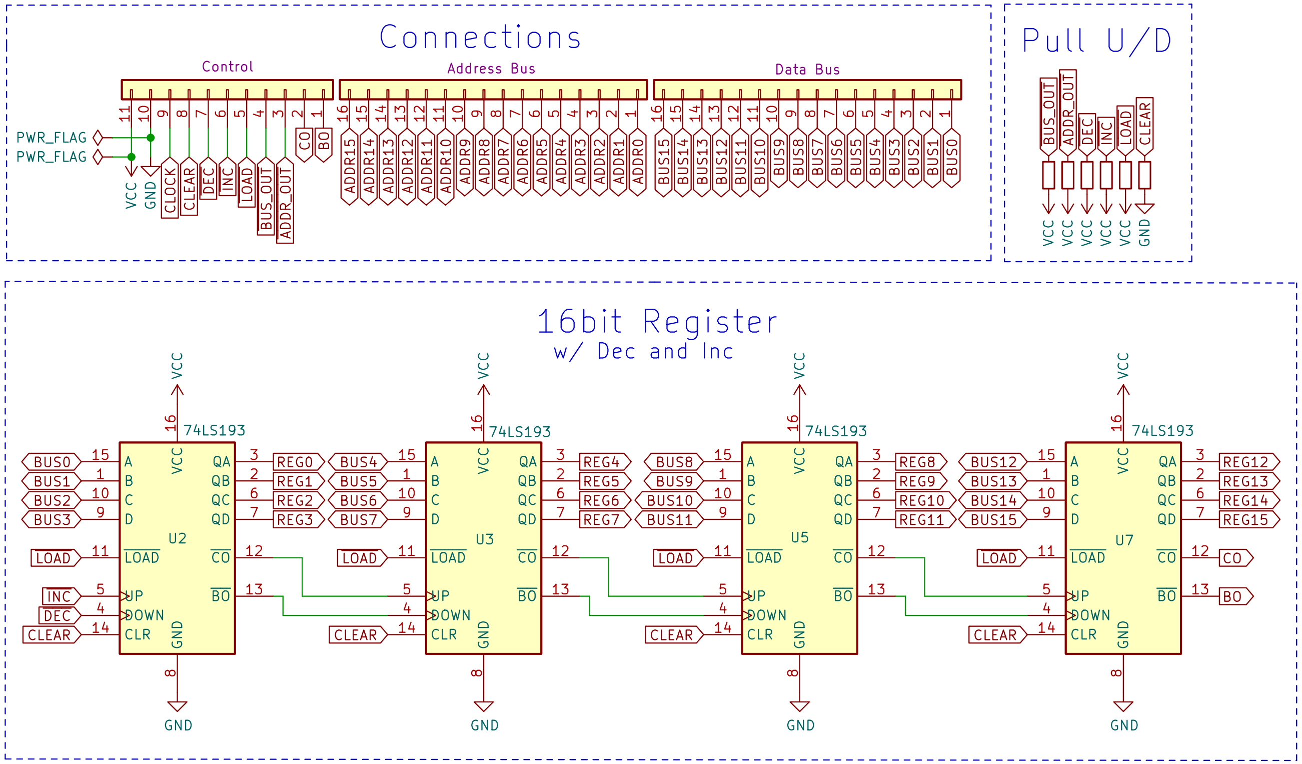
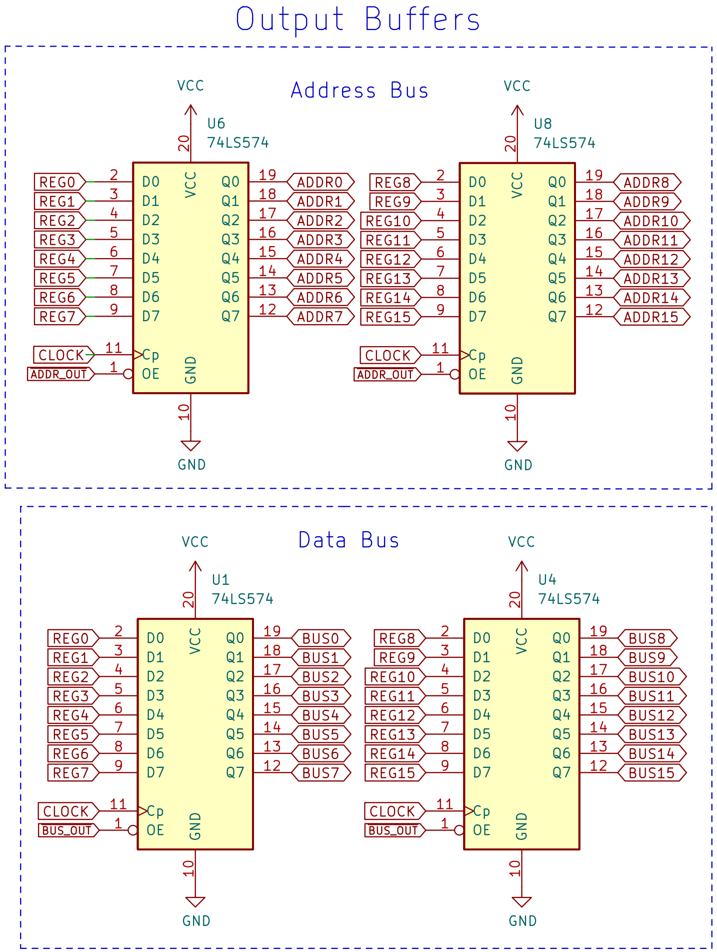
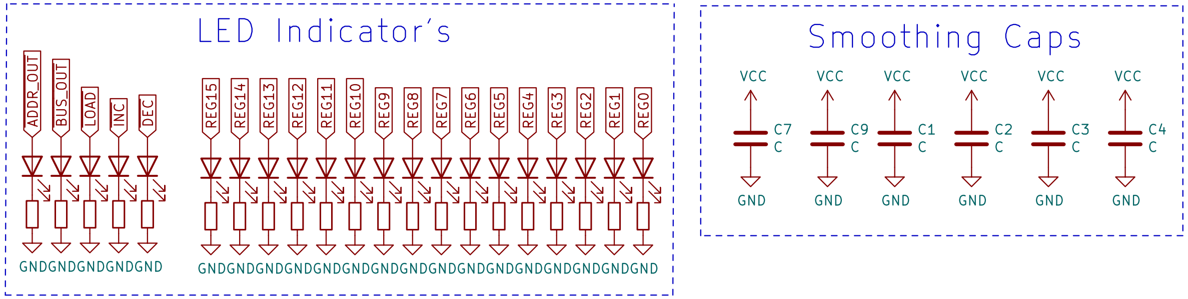
16bit Address Register

These registers provide functionality to different components such as the Stack Pointer, Program Counter, or any 16bit GPR. They have the ability to INC or DEC without using the ALU.

CLK	$\overline{DEC}$	$\overline{INC}$	$\overline{LOAD}$	$\overline{BUS}$	$\overline{ADDR}$		
x	H	H	H	H	H	–	NOOP
/	H	H	L	x	x	–	Load
/	L	H	x	x	x	–	Decrement
/	H	L	x	x	x	–	Increment







Sheet: /
File: address-register-smd.kicad_sch

Title: 16 Bit Address Register w/ INC and DEC

Size: A4 Date: **Rev: 3**
KiCad E.D.A. kicad (6.0.0-0) Id: 1/1

Arithmetic Module

8bit Arithmetic Module provides ADD, ADC, SUB, SBB, INC, DEC

DEC A: MUX_SEL: 1, TC: 0, CI: 0

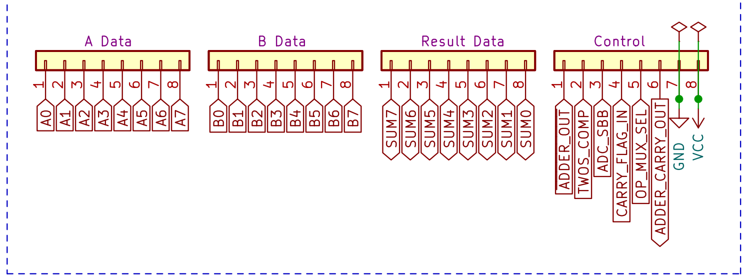
A - B: MUX_SEL: 0, TC: 1, CI: 1

A + B: MUX_SEL: 0, TC: 0, CI: 0

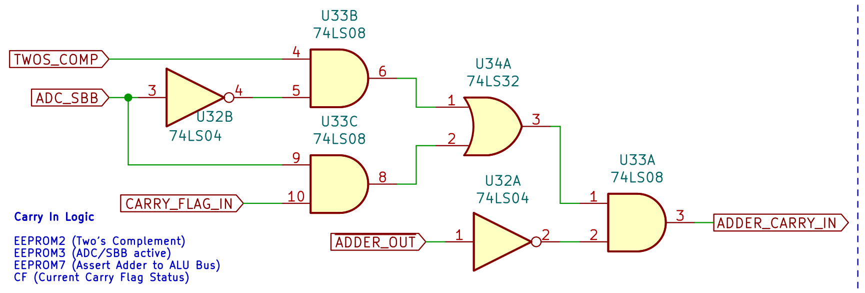
INC A: MUX_SEL: 1, TC: 1, CI: 1

A - B - Ci: MUX_SEL: 0, TC: 1, CI: ?
A + B + Cf: MUX_SEL: 0, TC: 0, CI: ?

Connectors



Arithmetic Carry In Logic

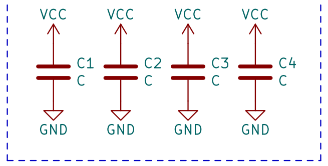


Carry In Logic

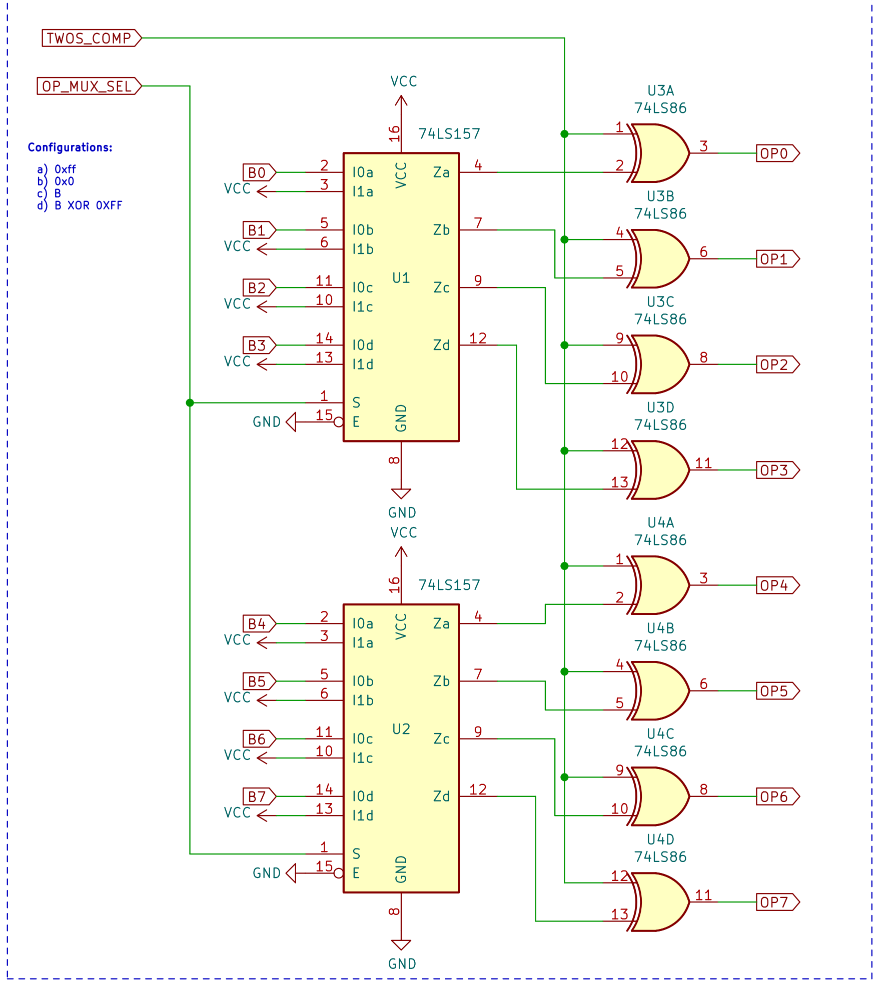
EEPROM2 (Two's Complement)
EEPROM3 (ADC/SBB active)
EEPROM7 (Assert Adder to ALU Bus)
CF (Current Carry Flag Status)

a) If twos complement, or
b) if ADC or SBB, and current carry flag status is active

Smoothing Caps



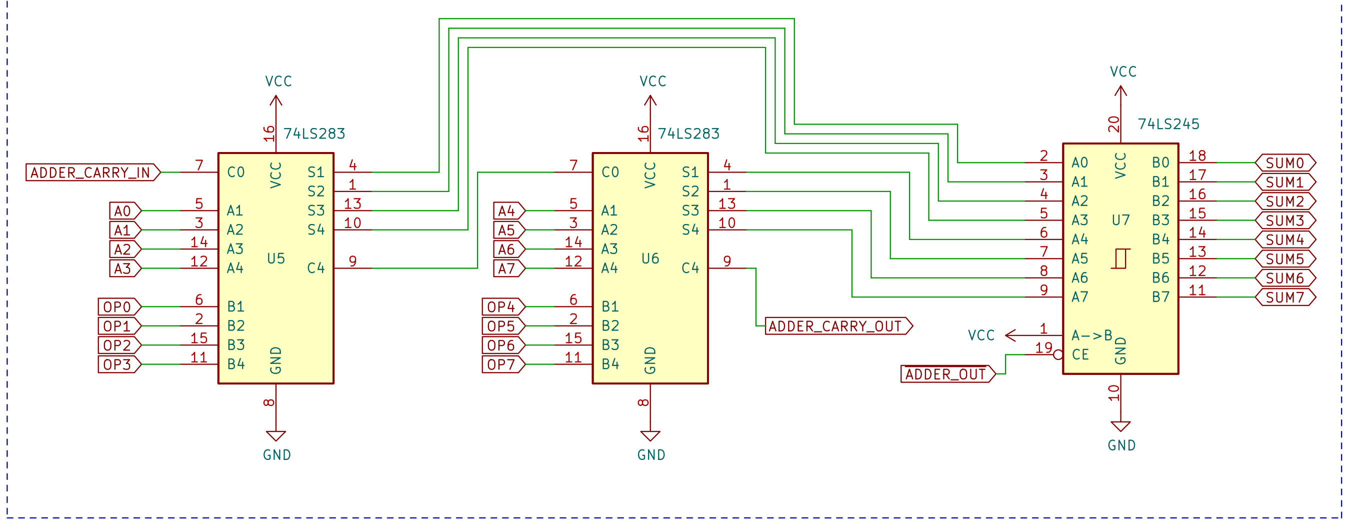
Operand Multiplexer



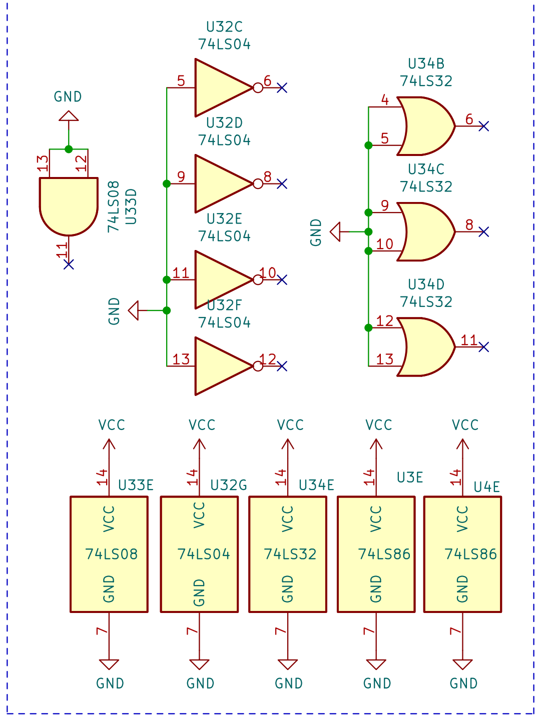
Configurations:

a) 0xff
b) 0x0
c) B
d) B XOR 0XFF

Full Adder w/ Carry



Logic Power



ADD / SUB / ADC / SBB / INC / DEC

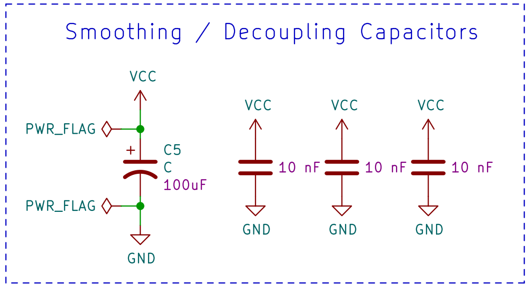
Sheet: /
File: Arithmetic.kicad_sch

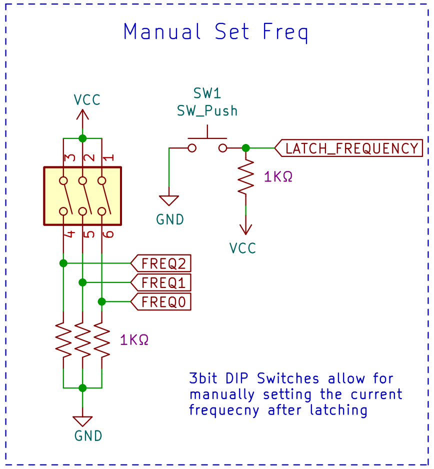
Title: Arithmetic Module

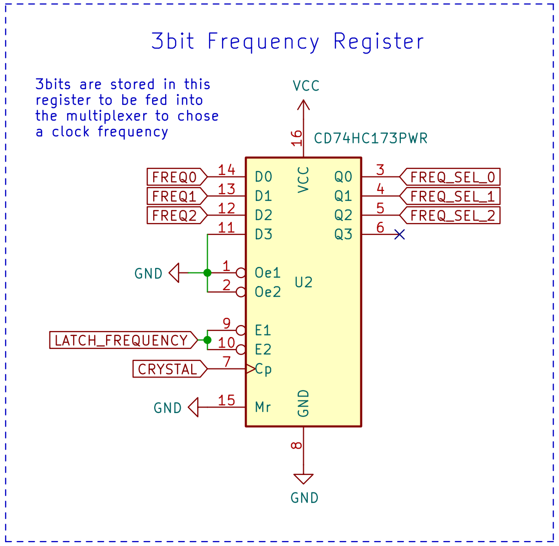
Size: A3   Date: **Rev: 3**
KiCad E.D.A. kicad (6.0.0-0)   Id: 1/1

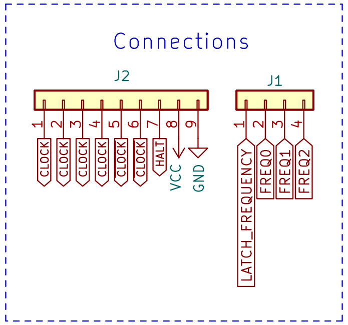
Clock

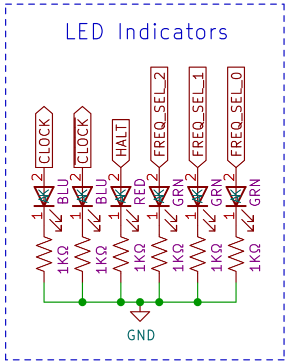
Adjustable frequency driven by a full can crystal.
Toggle Mode from Auto to Manual Pulse.

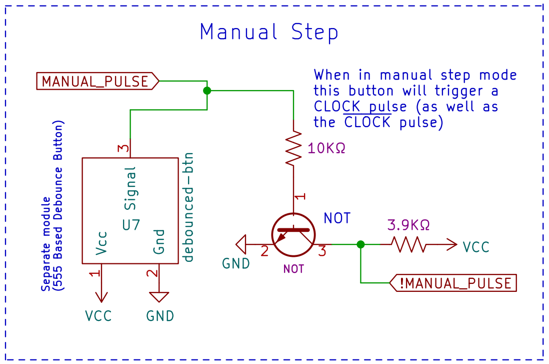


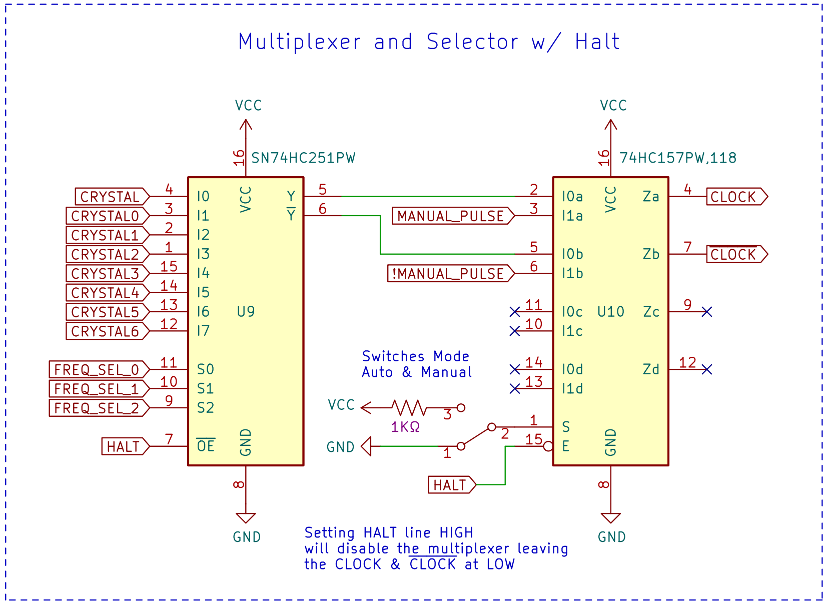

Manual Set Freq
VCC
SW1
SW_Push
LATCH_FREQUENCY
1KΩ
GND
VCC
FREQ2
FREQ1
FREQ0
1KΩ
GND
3bit DIP Switches allow for manually setting the current frequecny after latching

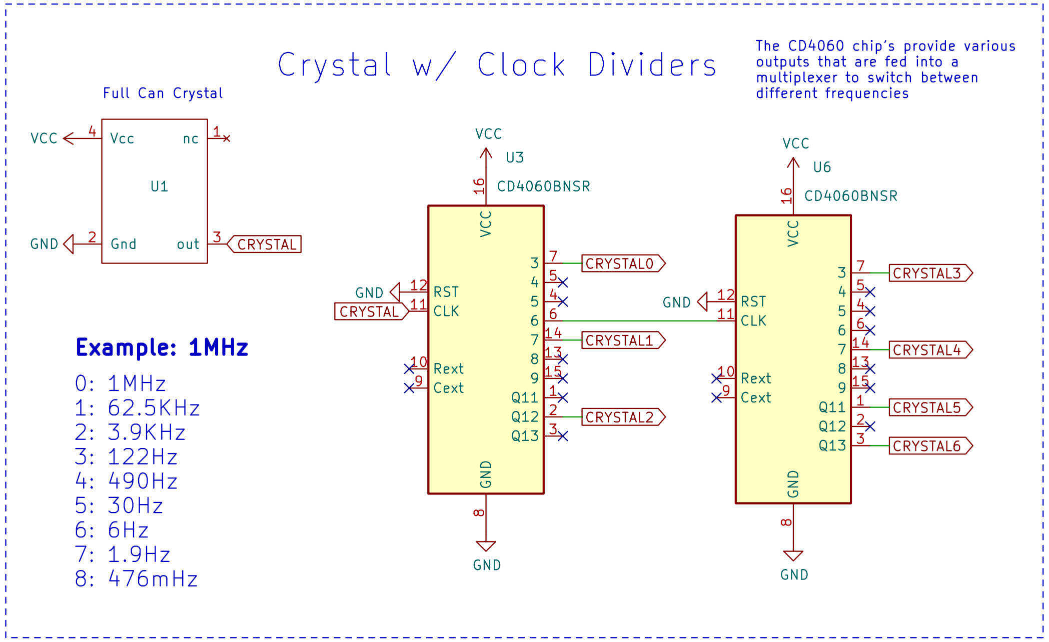







Manual Step
MANUAL_PULSE
When in manual step mode this button will trigger a CLOCK pulse (as well as the CLOCK pulse)
Separate module (555 Based Debounce Button)
Signal
U7
Vcc
Gnd
debounced-btn
VCC
GND
10KΩ
NOT
GND
NOT
3.9KΩ
VCC
!MANUAL_PULSE





Example: 1MHz

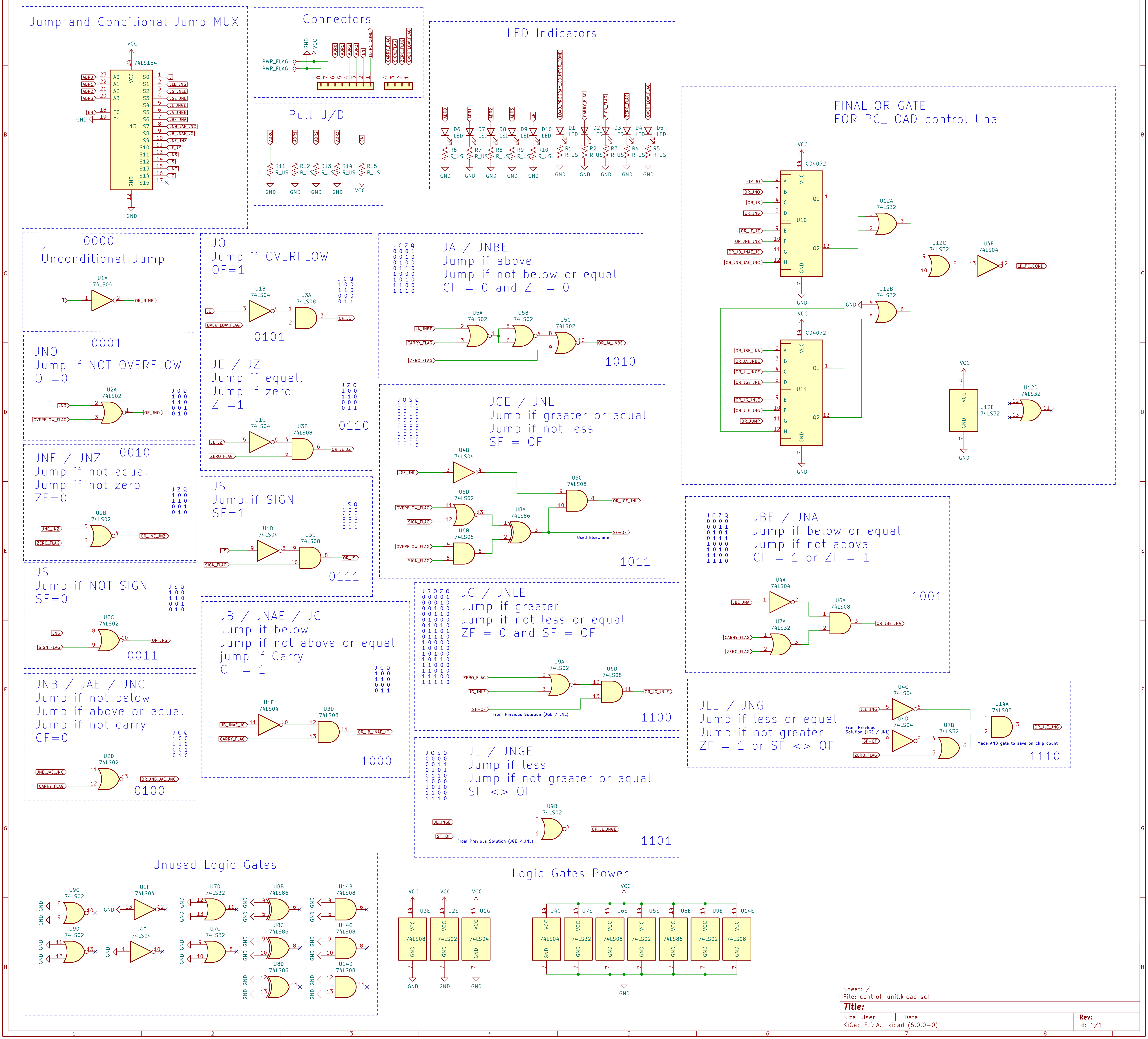
0: 1MHz
1: 62.5KHz
2: 3.9KHz
3: 122Hz
4: 490Hz
5: 30Hz
6: 6Hz
7: 1.9Hz
8: 476mHz

Programable with Manual Step
theWickedWebDev/8-bit-computer
Sheet: /
File: clockv2.kicad_sch

Title: Clock Module

Size: User Date: **Rev: v2**
KiCad E.D.A. kicad (6.0.0-0) Id: 1/1

ALU Conditional Jump Logic


Jump and Conditional Jump MUX
VCC
74LS154
U13
A0 A1 A2 A3 E0 E1 VCC GND
ADR0 ADR1 ADR2 ADR3 EN GND
S0 J
S1 JLE_JNG
S2 JG_JNLE
S3 JGE_JNL
S4 JL_JNGE
S5 JA_JNBE
S6 JBE_JNA
S7 JNB_JAE_JNC
S8 JB_JNAE_JC
S9 JNE_JNZ
S10 JE_JZ
S11 JNS
S12 JS
S13 JNO
S14 JO
S15
GND
Connectors
PWR_FLAG
PWR_FLAG
GND VCC ADR0 ADR1 ADR2 ADR3 EN LD_PC_COND
CARRY_FLAG SIGN_FLAG ZERO_FLAG OVERFLOW_FLAG
Pull U/D
ADR0 ADR1 ADR2 ADR3 EN
R11 R_US R12 R_US R13 R_US R14 R_US R15 R_US
GND GND GND GND VCC
LED Indicators
ADR0 ADR1 ADR2 ADR3 EN LOAD_PROGRAM_COUNTER_COND CARRY_FLAG SIGN_FLAG ZERO_FLAG OVERFLOW_FLAG
D6 LED D7 LED D8 LED D9 LED D10 LED D1 LED D2 LED D3 LED D4 LED D5 LED
R6 R_US R7 R_US R8 R_US R9 R_US R10 R_US R1 R_US R2 R_US R3 R_US R4 R_US R5 R_US
GND
FINAL OR GATE
FOR PC_LOAD control line
VCC
CD4072
U10
OR_JO OR_JNO OR_JS OR_JNS OR_JE_JZ OR_JNE_JNZ OR_JB_JNAE_JC OR_JNB_JAE_JNC
A B C D E F G H Q1 Q2
U11
OR_JBE_JNA OR_JA_JNBE OR_JL_JNGE OR_JGE_JNL OR_JG_JNLE OR_JLE_JNG OR_JUMP
U12A 74LS32
U12B 74LS32
U12C 74LS32
U4F 74LS04
LD_PC_COND
U12E 74LS32
U12D 74LS32
GND
U1A 74LS04 J OR_JUMP
U1B 74LS04 U3A 74LS08 JO OVERFLOW_FLAG OR_JO
U2A 74LS02 JNO OVERFLOW_FLAG OR_JNO
U1C 74LS04 U3B 74LS08 JE_JZ ZERO_FLAG OR_JE_JZ
U2B 74LS02 JNE_JNZ ZERO_FLAG OR_JNE_JNZ
U1D 74LS04 U3C 74LS08 JS SIGN_FLAG OR_JS
U2C 74LS02 JNS SIGN_FLAG OR_JNS
U1E 74LS04 U3D 74LS08 JB_JNAE_JC CARRY_FLAG OR_JB_JNAE_JC
U2D 74LS02 JNB_JAE_JNC CARRY_FLAG OR_JNB_JAE_JNC
U5A 74LS02 U5B 74LS02 U5C 74LS02 JA_JNBE CARRY_FLAG ZERO_FLAG OR_JA_JNBE
U4B 74LS04 U5D 74LS02 U6B 74LS08 U8A 74LS86 U6C 74LS08
JGE_JNL OVERFLOW_FLAG SIGN_FLAG OR_JGE_JNL SF=OF
Used Elsewhere
U9A 74LS02 U6D 74LS08 ZERO_FLAG JG_JNLE SF=OF OR_JG_JNLE
From Previous Solution (JGE / JNL)
U9B 74LS02 JL_JNGE SF=OF OR_JL_JNGE
From Previous Solution (JGE / JNL)
U4A 74LS04 U7A 74LS32 U6A 74LS08 JBE_JNA CARRY_FLAG ZERO_FLAG OR_JBE_JNA
U4C 74LS04 U4D 74LS04 U7B 74LS32 U14A 74LS08 JLE_JNG SF=OF ZERO_FLAG OR_JLE_JNG
From Previous Solution (JGE / JNL)
Made AND gate to save on chip count
Unused Logic Gates
U9C 74LS02 U9D 74LS02 U1F 74LS04 U4E 74LS04 U7D 74LS32 U7C 74LS32 U8B 74LS86 U8C 74LS86 U8D 74LS86 U14B 74LS08 U14C 74LS08 U14D 74LS08
GND
Logic Gates Power
VCC
U3E 74LS08 U2E 74LS02 U1G 74LS04
U4G 74LS04 U7E 74LS32 U6E 74LS08 U5E 74LS02 U8E 74LS86 U9E 74LS02 U14E 74LS08
GND

J 0000
Unconditional Jump

JO
Jump if OVERFLOW
OF=1

J	O	Q
1	0	0
1	1	0
0	0	0
0	1	1

0101

0001
JNO
Jump if NOT OVERFLOW
OF=0

J	O	Q
1	0	0
1	1	0
0	0	1
0	1	0

JE / JZ
Jump if equal,
Jump if zero
ZF=1

J	Z	Q
1	0	0
1	1	0
0	0	0
0	1	1

0110

JNE / JNZ 0010
Jump if not equal
Jump if not zero
ZF=0

J	Z	Q
1	0	0
1	1	0
0	0	1
0	1	0

JS
Jump if SIGN
SF=1

J	S	Q
1	0	0
1	1	0
0	0	0
0	1	1

0111

JS
Jump if NOT SIGN
SF=0

J	S	Q
1	0	0
1	1	0
0	0	1
0	1	0

0011

JNB / JAE / JNC
Jump if not below
Jump if above or equal
Jump if not carry
CF=0

J	C	Q
1	0	0
1	1	0
0	0	1
0	1	0

0100

JB / JNAE / JC
Jump if below
Jump if not above or equal
jump if Carry
CF = 1

J	C	Q
1	0	0
1	1	0
0	0	0
0	1	1

1000

JA / JNBE
Jump if above
Jump if not below or equal
CF = 0 and ZF = 0

J	C	Z	Q
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

1010

JGE / JNL
Jump if greater or equal
Jump if not less
SF = OF

J	O	S	Q
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

1011

JG / JNLE
Jump if greater
Jump if not less or equal
ZF = 0 and SF = OF

J	S	O	Z	Q
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

1100

JL / JNGE
Jump if less
Jump if not greater or equal
SF <> OF

J	O	S	Q
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

1101

JBE / JNA
Jump if below or equal
Jump if not above
CF = 1 or ZF = 1

J	C	Z	Q
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

1001

JLE / JNG
Jump if less or equal
Jump if not greater
ZF = 1 or SF <> OF

1110

Sheet: /
File: control-unit.kicad_sch

Title:

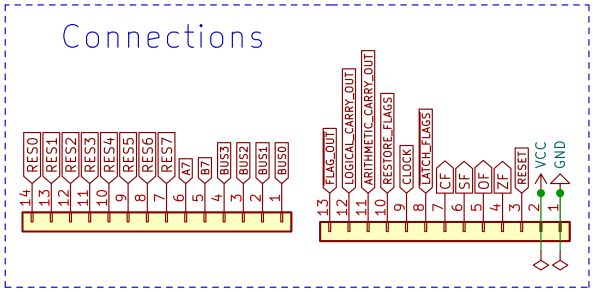
Size: User Date: Rev:
KiCad E.D.A. kicad (6.0.0-0) Id: 1/1

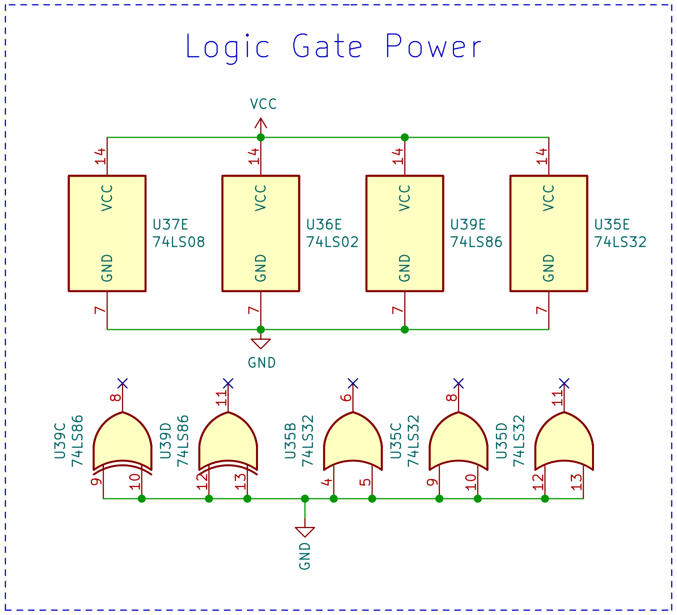
FLAGS REGISTER

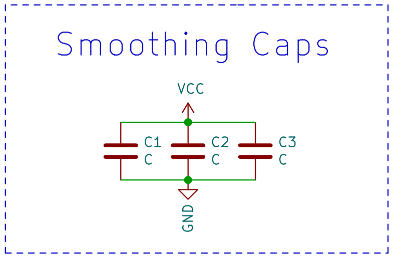
LATCH_FLAGS – A LOW signal will store the data asserted from the multiplexer into the Flags Register (FR)

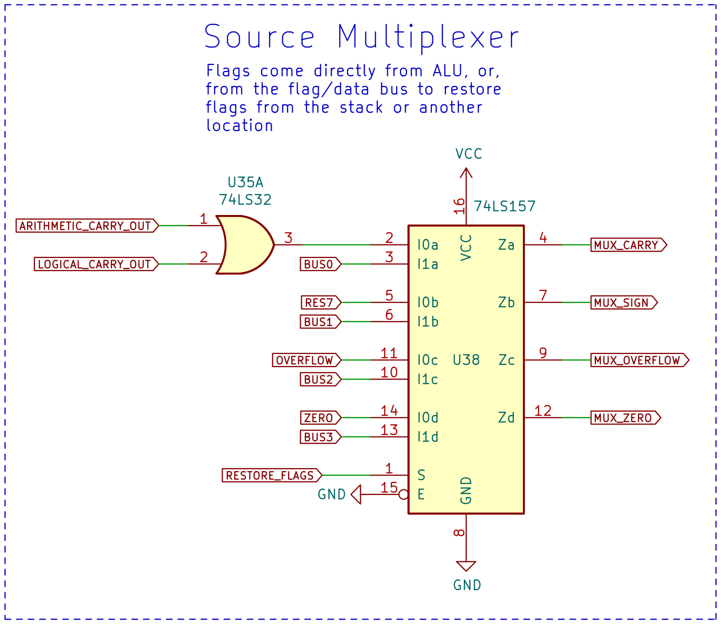
- RESTORE: LOW, uses signals from ALU
- RESTORE: HIGH, uses signal asserted on data bus

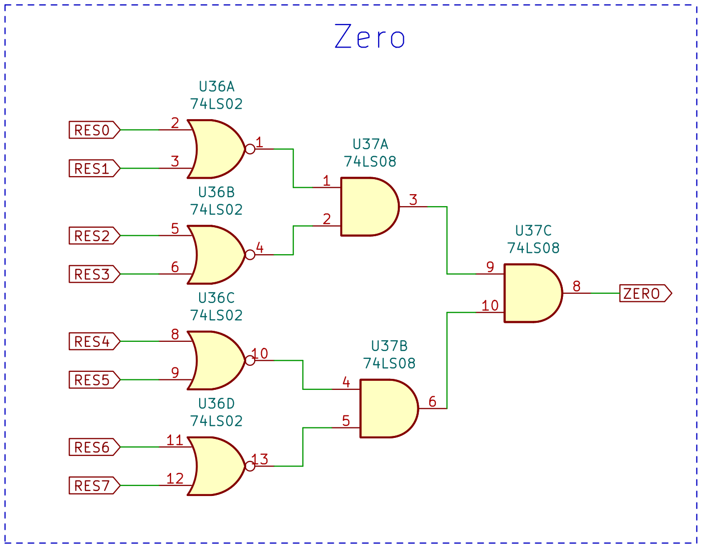
FLAG_OUT – Assers the current flags statuses onto the Data bus, typically used to push it onto the stack to handle an ISR

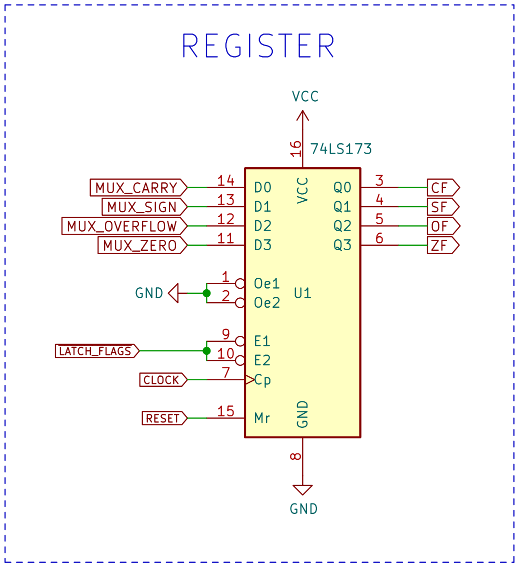


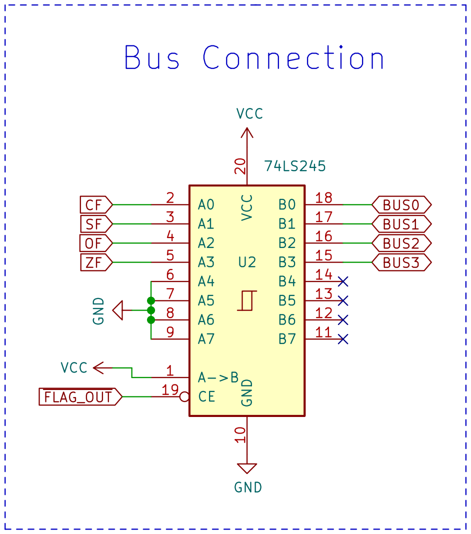


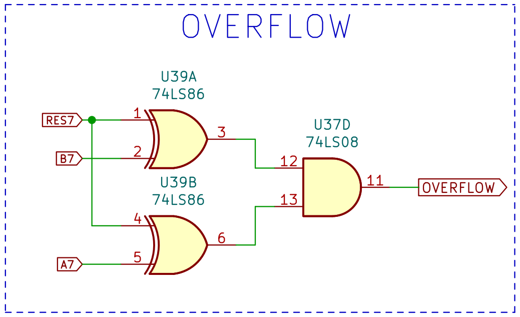


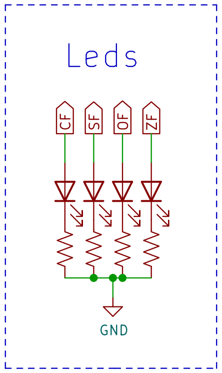












For storing and asserting current flag statuses from ALU

theWickedWebDev/8-Bit-Computer

Sheet: /

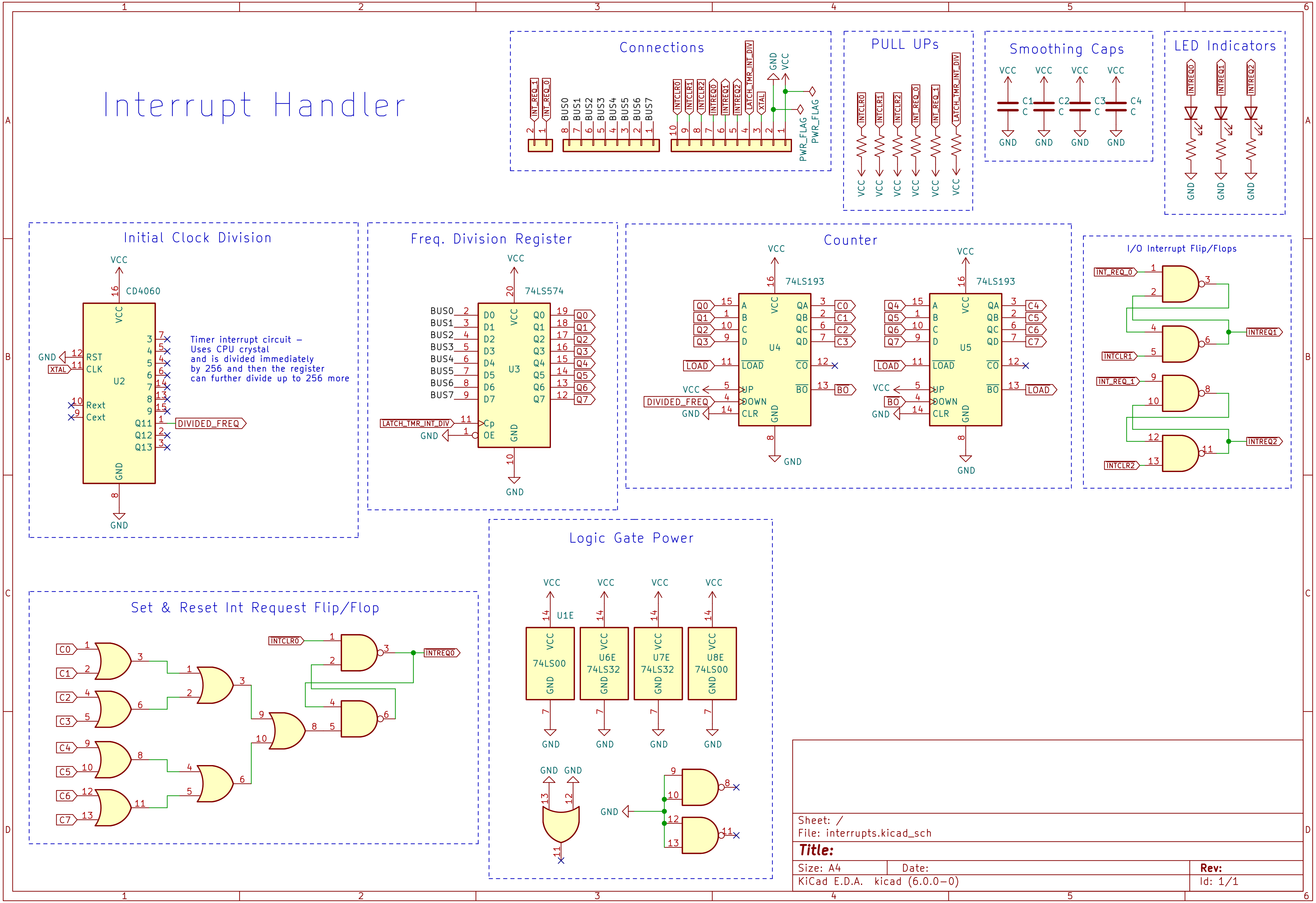
File: Flags Register.kicad_sch

Title: Flags Register

Size: User Date: 2022-01-03 **Rev: 3**

KiCad E.D.A. kicad (6.0.0-0) Id: 1/1

Interrupt Handler



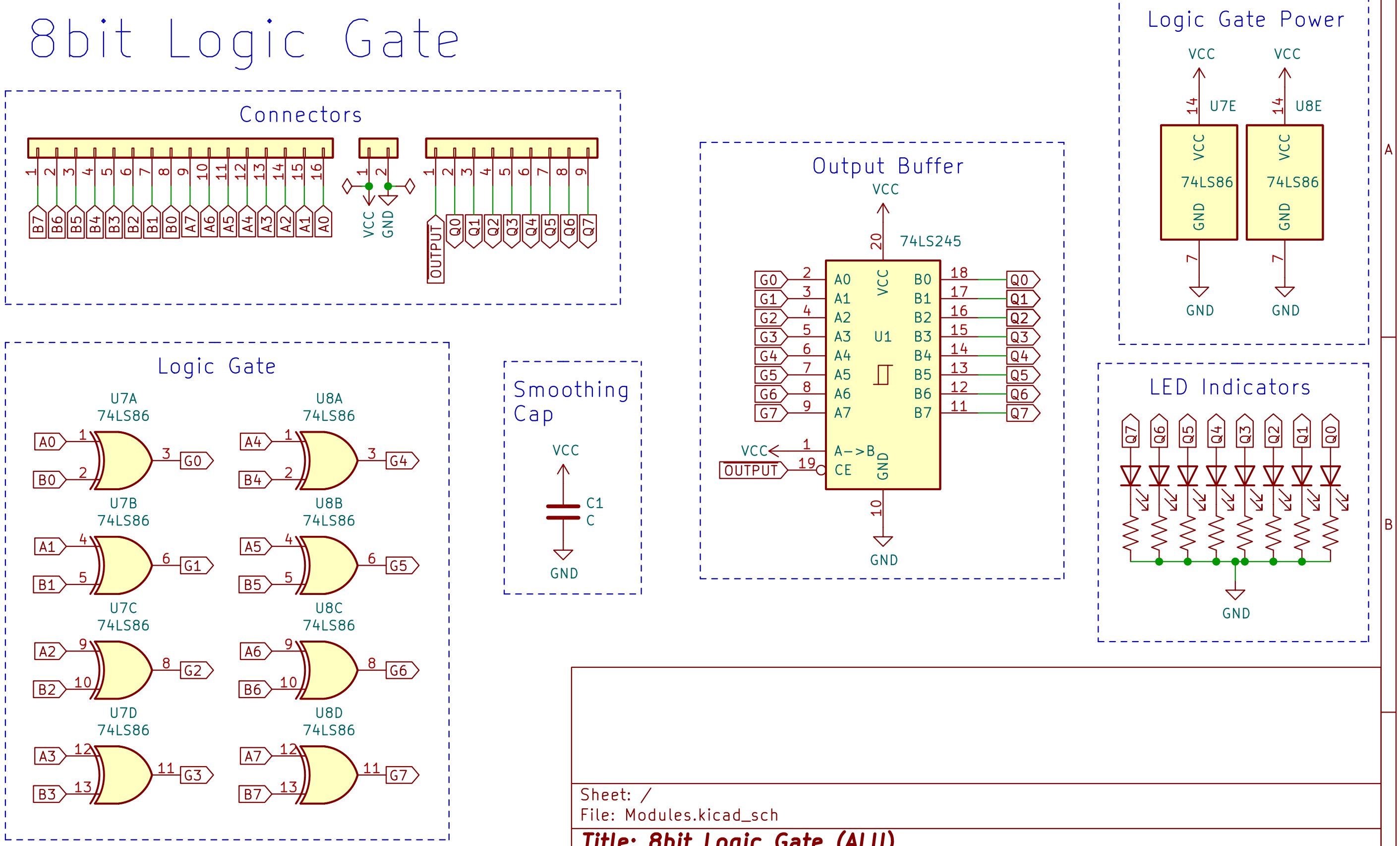
Timer interrupt circuit – Uses CPU crystal and is divided immediately by 256 and then the register can further divide up to 256 more

Sheet: /
File: interrupts.kicad_sch

Title:

Size: A4 | Date: | **Rev:**

KiCad E.D.A. kicad (6.0.0-0) | Id: 1/1



Sheet: /

File: Modules.kicad_sch

Title: 8bit Logic Gate (ALU)

Size: A5 Date: **Rev: 3**

KiCad E.D.A. kicad (6.0.0-0) Id: 1/1

Memory Module

Provides 992K Of RAM and 32K Of ROM. Address space from $0:$7fff is reserved from ROM and $8000:$fffff is reserved for RAM.

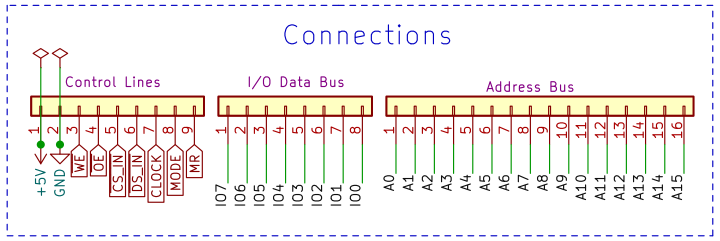
The code segment (CS) and the data segment(DS) registers should be initialized at the start of your program.

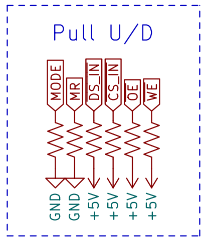
Setting DS_IN or CS_IN low will latch the value presented on the data bus into the corresponding segment register.

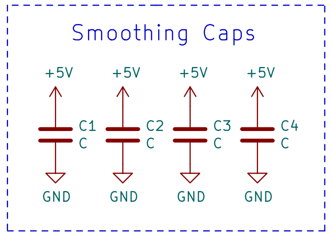
A HIGH signal on the WE line will write to RAM as long as the address provided falls into a valid RAM address.

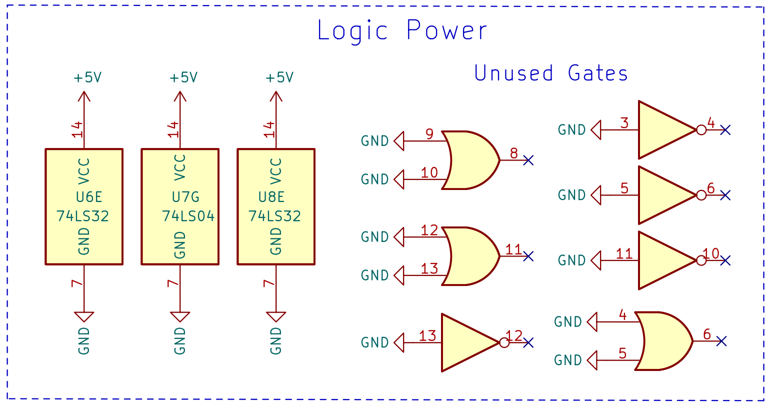
A LOW signal on the OE line will assert the contents at the specified address out onto the data bus.

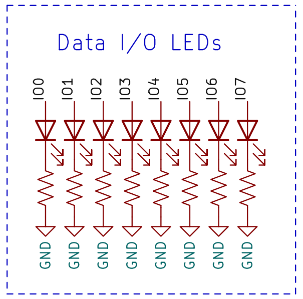
MODE selects which segment to use, Code or Data. A HIGH signal will retrieve DATA back from memory by using the DS, whereas a LOW signal will return back CODE

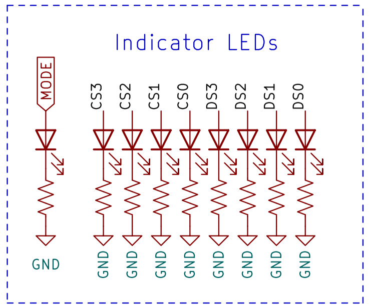

Connections

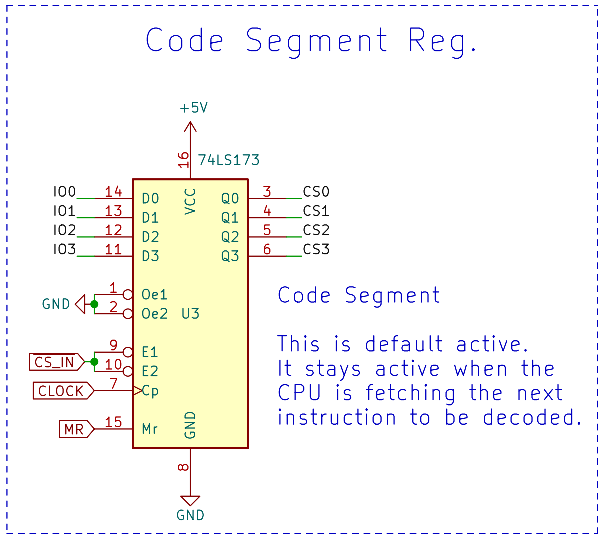

Pull U/D


Smoothing Caps


Logic Power

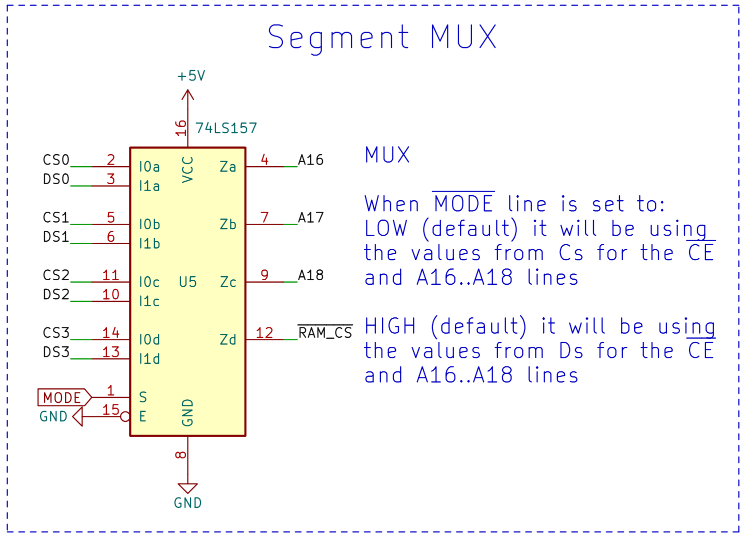

Data I/O LEDs


Indicator LEDs


Code Segment Reg.

Code Segment

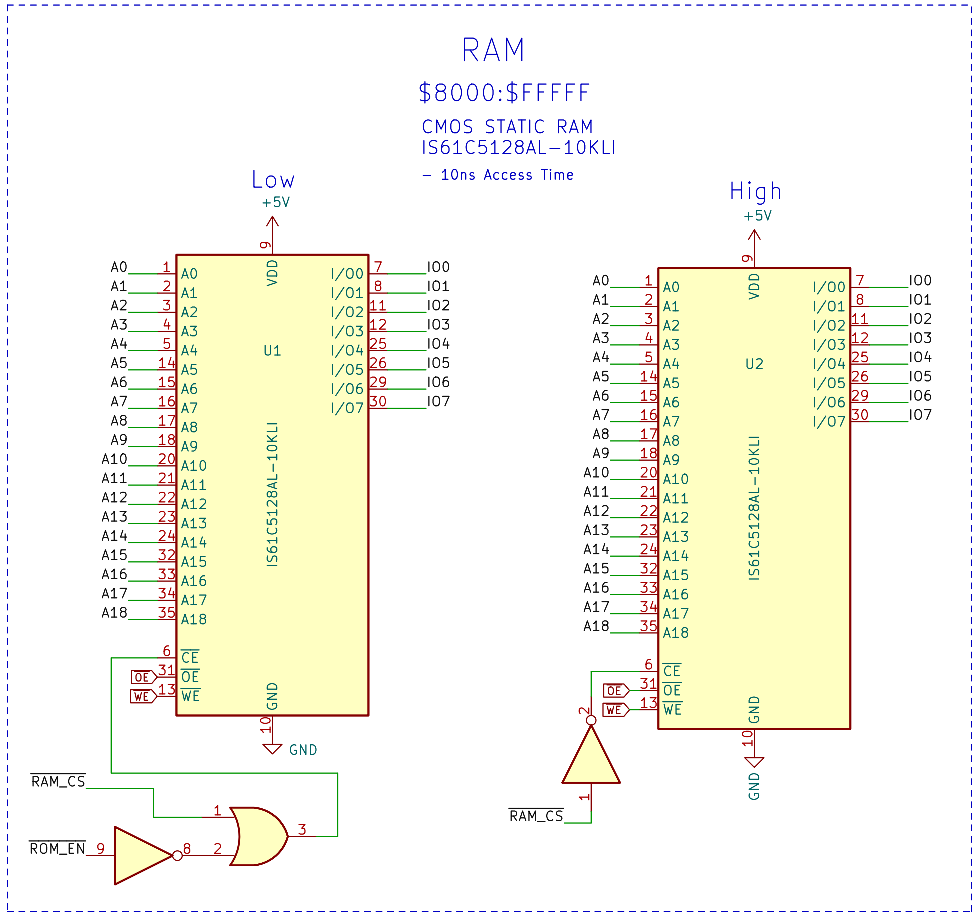
This is default active.
It stays active when the CPU is fetching the next instruction to be decoded.

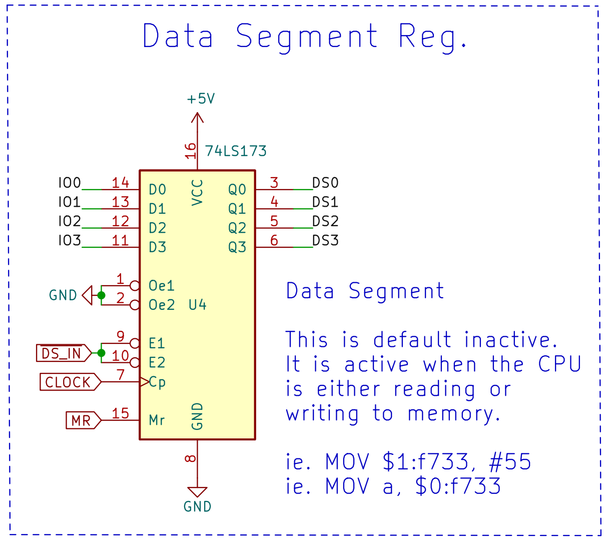

Segment MUX

MUX

When MODE line is set to:
LOW (default) it will be using the values from Cs for the CE and A16..A18 lines

HIGH (default) it will be using the values from Ds for the CE and A16..A18 lines

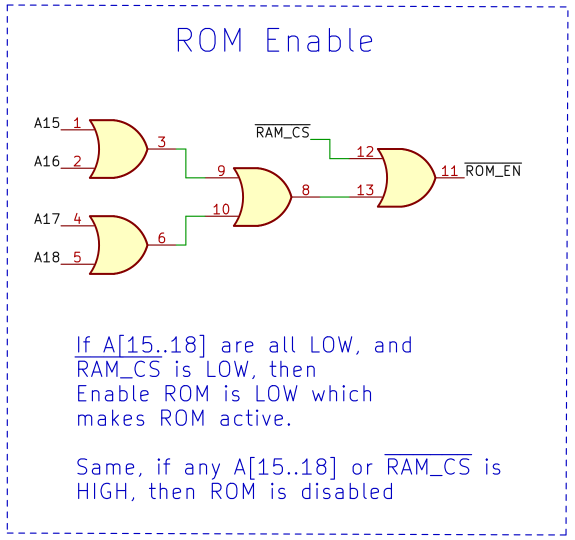

RAM
$8000:$FFFFF
CMOS STATIC RAM
IS61C5128AL-10KLI
- 10ns Access Time


Data Segment Reg.

Data Segment

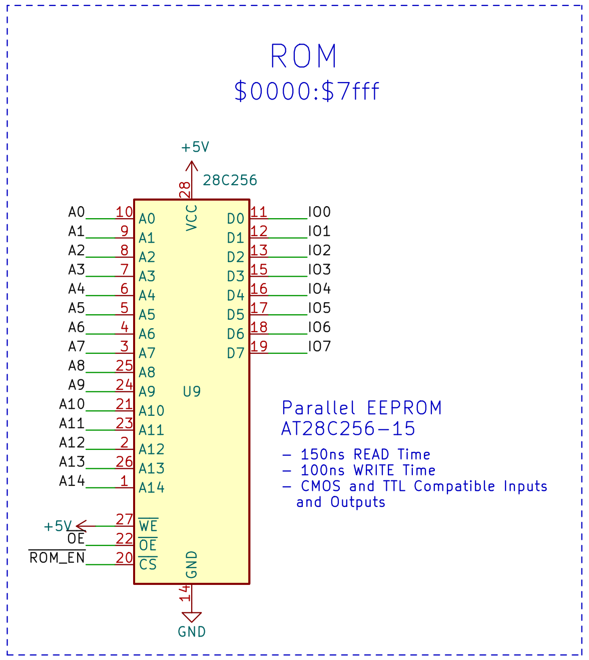
This is default inactive.
It is active when the CPU is either reading or writing to memory.

ie. MOV $1:f733, #55
ie. MOV a, $0:f733


ROM Enable

If A[15..18] are all LOW, and RAM_CS is LOW, then Enable ROM is LOW which makes ROM active.

Same, if any A[15..18] or RAM_CS is HIGH, then ROM is disabled


ROM
$0000:$7fff

Parallel EEPROM
AT28C256-15

- 150ns READ Time
- 100ns WRITE Time
- CMOS and TTL Compatible Inputs and Outputs

Sheet: /
File: RAM-MODULE.kicad_sch

Title: Memory Module

Size: A3	Date: 2021-12-02	Rev: 3
KiCad E.D.A. kicad (6.0.0-0)		Id: 1/1

GPR Assertions Controller

CONTROL WORD HIGH BYTE

Assertions to 8bit Bus

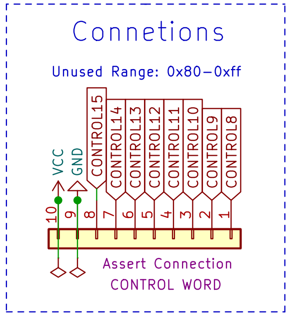
```
11xxxxxx - Assertions NOOP
00xx0000 - Assert A
00xx0001 - Assert C
00xx0010 - Assert D
00xx0011 - Assert E
00xx0100 - Assert Scratch 1
00xx0101 - Assert Scratch 2
00xx0110 - Assert Output 1
00xx0111 - Assert Output 2
00xx1000 - Assert Output 3
00xx1001 - Assert TX LSB
00xx1010 - Assert TX MSB
00xx1011 -     **** unused active low
00xx1100 -     **** unused active low
00xx1101 -     **** unused active low
00xx1110 -     **** unused active low
00xx1111 -     **** unused active low
```

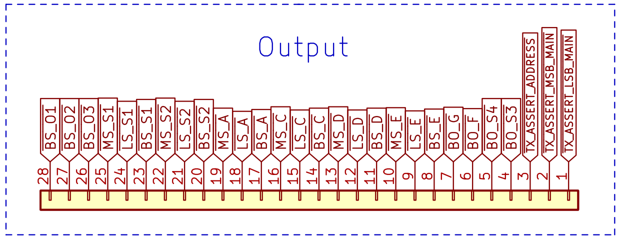
Assertions to 16bit Address Bus

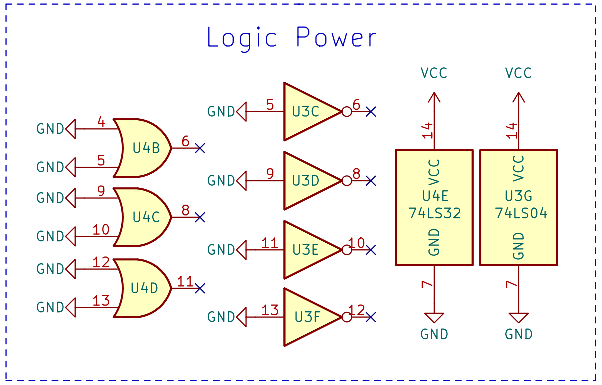
```
01xxx000 - Assert F
01xxx001 - Assert G
01xxx010 - Assert S3
01xxx011 - Assert S4
01xxx100 - Assert TX
01xxx101 - Assert r onto 16bit bus**
01xxx110 - Assert rr onto 16bit bus**
01xxx111 -     **** unused active low
```

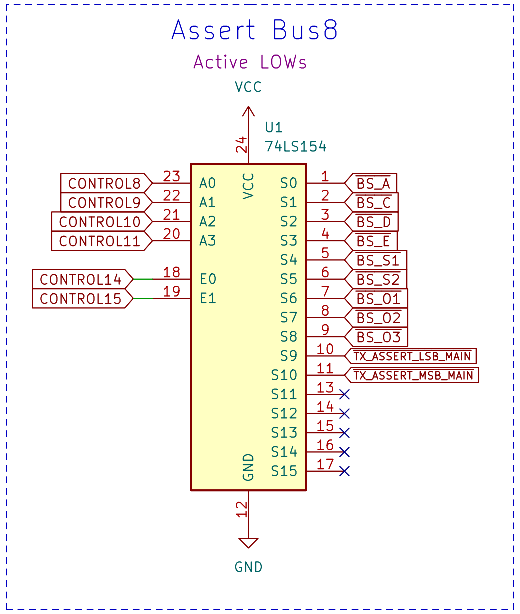
```
01000101 - Assert Aonto 16bit bus
01001101 - Assert C onto 16bit bus
01010101 - Assert D onto 16bit bus
01011101 - Assert E onto 16bit bus
01100101 - Assert S1 onto 16bit bus
01101101 - Assert S2 onto 16bit bus
01110101 -     ****unused r => 16bit LSB
01111101 -     ****unused r => 16bit LSB
```

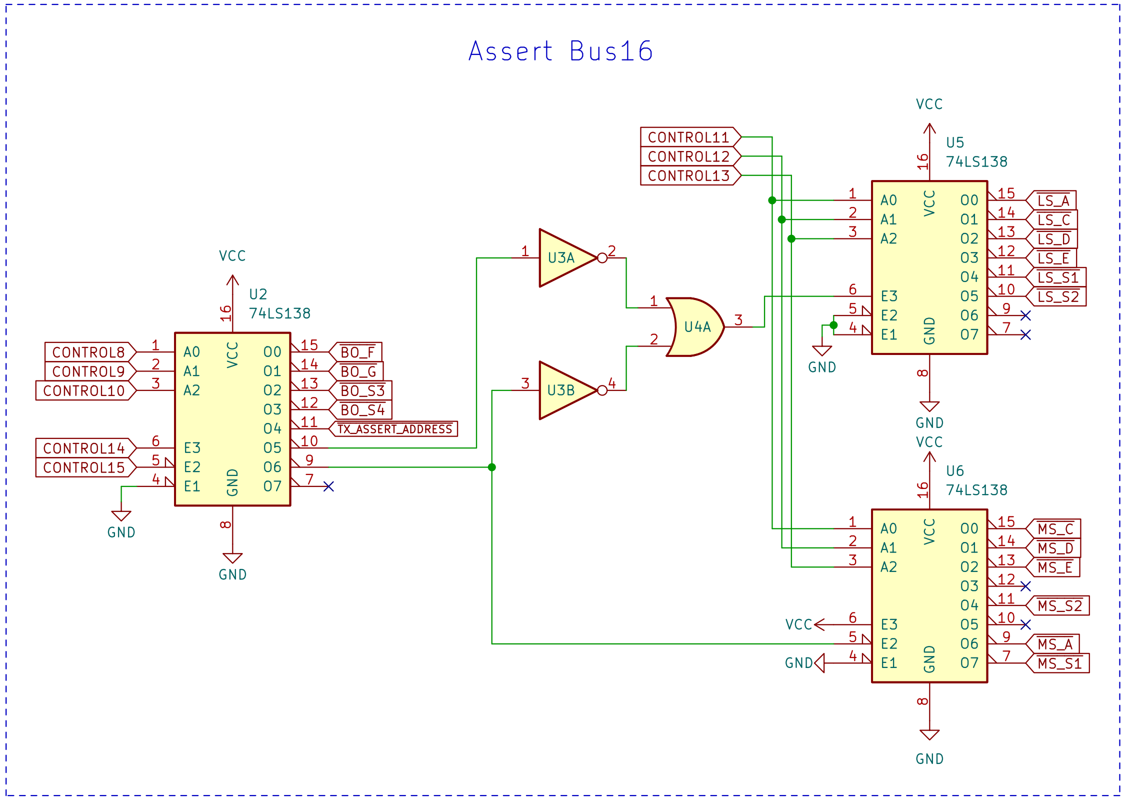
```
01000110 - Assert AC onto 16bit bus
01001110 - Assert CD onto 16bit bus
01010110 - Assert DE onto 16bit bus
01011110 -     ****unused rr => 16bit (Assert E onto 16bit bus)
01100110 - Assert SS onto 16bit bus
01101110 -     ****unused rr => 16bit (Assert S2 onto 16bit bus)
01110110 - Assert A onto MSB of 16bit bus
01111110 - Assert S1 onto MSB of 16bit bus
```

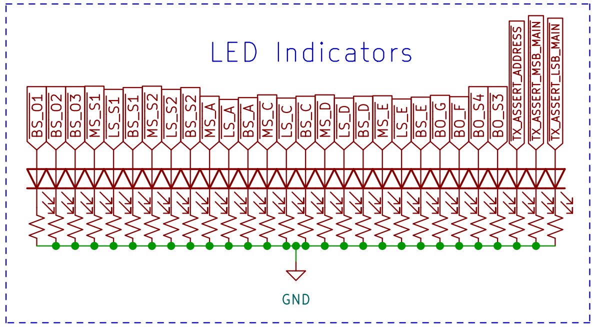



Output
BS_O1 BS_O2 BS_O3 MS_S1 LS_S1 BS_S1 MS_S2 LS_S2 BS_S2 MS_A LS_A BS_A MS_C LS_C BS_C MS_D LS_D BS_D MS_E LS_E BS_E BO_G BO_F BO_S4 BO_S3 TX_ASSERT_ADDRESS TX_ASSERT_MSB_MAIN TX_ASSERT_LSB_MAIN
28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1







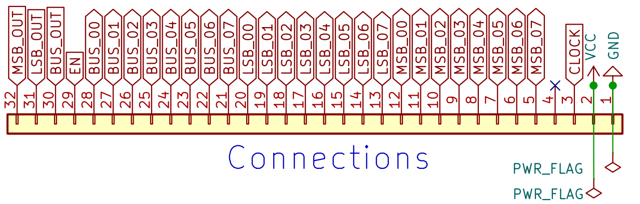

LED Indicators
BS_O1 BS_O2 BS_O3 MS_S1 LS_S1 BS_S1 MS_S2 LS_S2 BS_S2 MS_A LS_A BS_A MS_C LS_C BS_C MS_D LS_D BS_D MS_E LS_E BS_E BO_G BO_F BO_S4 BO_S3 TX_ASSERT_ADDRESS TX_ASSERT_MSB_MAIN TX_ASSERT_LSB_MAIN
GND

Sheet: /
File: register-board-controller.kicad_sch

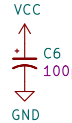
Title:

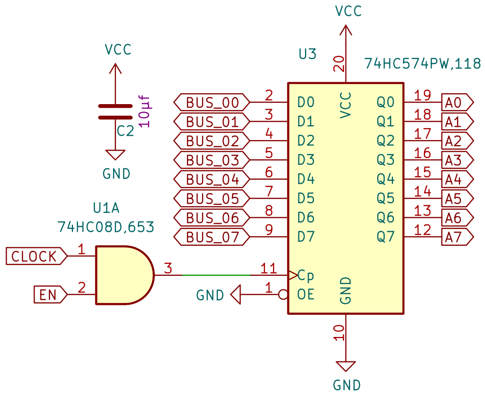
Size: User Date: **Rev:**
KiCad E.D.A. kicad (6.0.0-0) Id: 1/1

8bit General Purpose Register



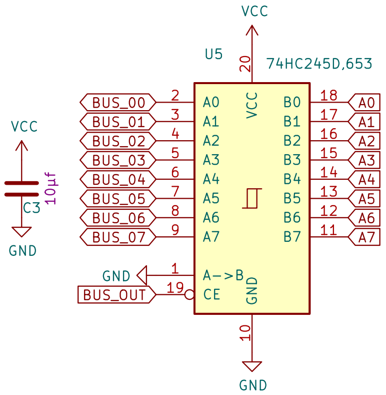
Connections





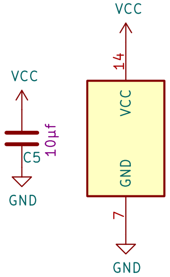
8bit Register

This will latch in data from the Data BUS with the rising edge of the clock and a HIGH signal on EN line



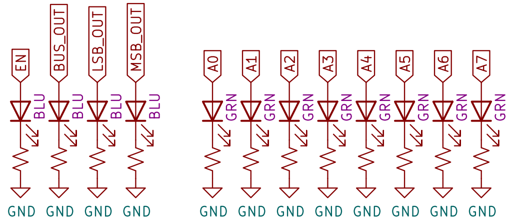
Non Inverting Buffer

When BUS_OUT is set LOW, this register will assert its value out onto the DATA BUS

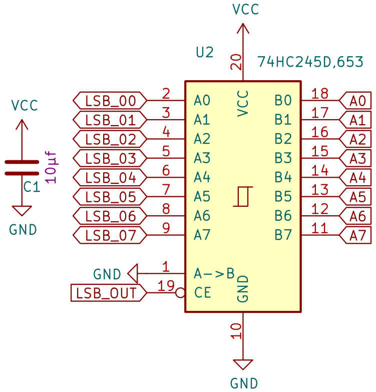


AND Gate

Used to Enable the Register along with the Clock Pulse

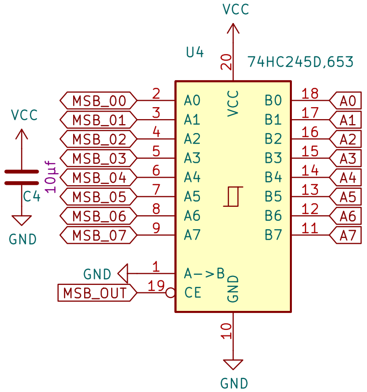


LED Indicators



Non Inverting Buffer

When LSB_OUT is set LOW, this register will assert its value out onto the LSB BUS



Non Inverting Buffer

When MSB_OUT is set LOW, this register will assert its value out onto the MSB BUS

Stores a byte of data and asserts it to two different busses.

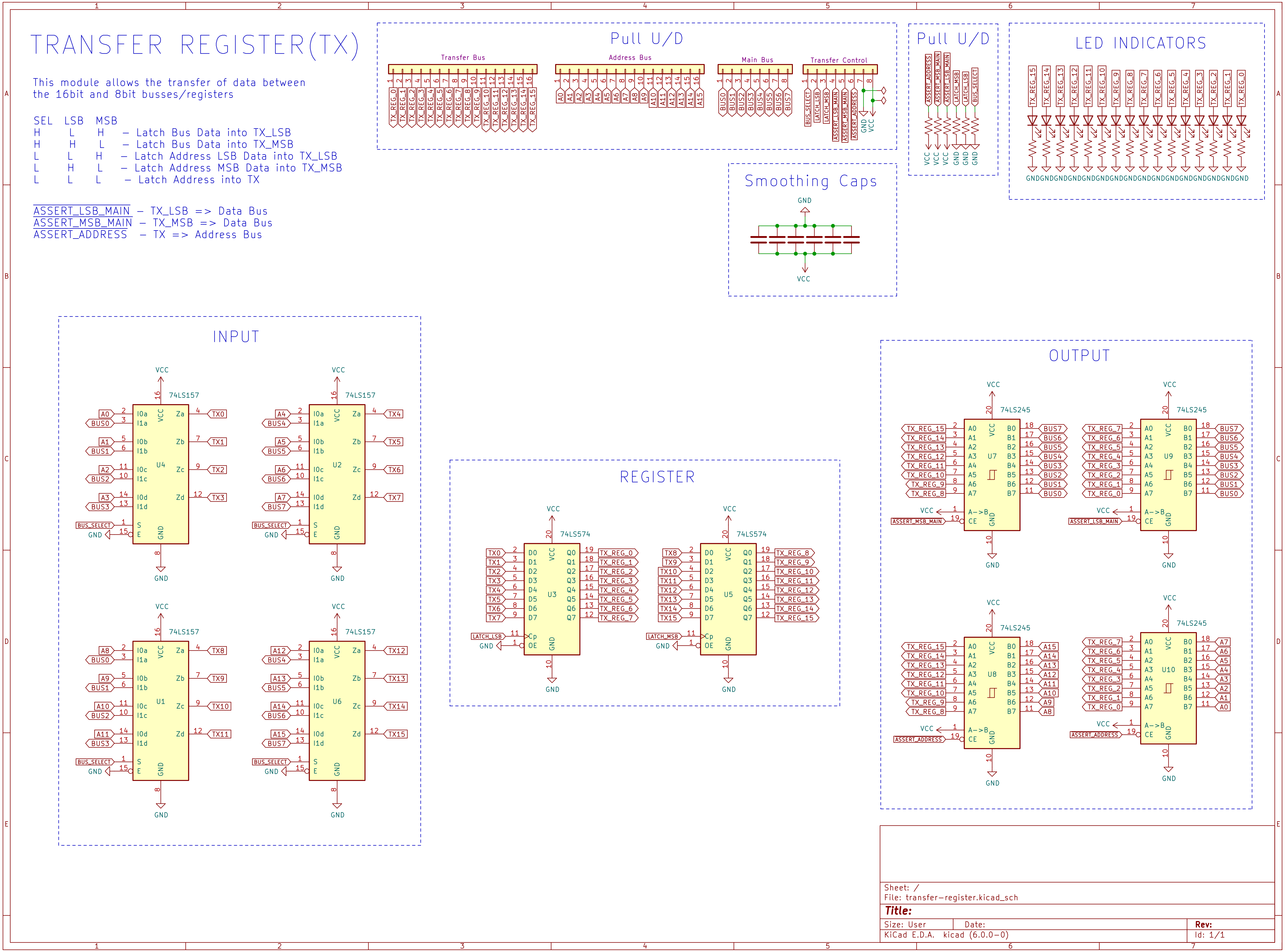
$\overline{\text{MSB\_OUT}}$	$\overline{\text{LSB\_OUT}}$	$\overline{\text{BUS\_OUT}}$	EN	CLK	
0	1	1	1	x	– Outputs to MSB of Address Bus
1	0	1	1	x	– Outputs to LSB of Address Bus
1	1	0	1	x	– Outputs to Data Bus
1	1	1	0	/	– Latches Data
1	1	1	1	x	– Noop

1 – HIGH
0 – LOW
x – Dont care
/ – Rising Edge

Sheet: /
File: smd–register–array.kicad_sch

Title: 8bit General Purpose Register

Size: A4	Date: 2021–08–02	**Rev: 3**
KiCad E.D.A. kicad (6.0.0–0)		Id: 1/1



TRANSFER REGISTER(TX)

This module allows the transfer of data between
the 16bit and 8bit busses/registers

SEL	LSB	MSB	
H	L	H	– Latch Bus Data into TX_LSB
H	H	L	– Latch Bus Data into TX_MSB
L	L	H	– Latch Address LSB Data into TX_LSB
L	H	L	– Latch Address MSB Data into TX_MSB
L	L	L	– Latch Address into TX

$\overline{\text{ASSERT\_LSB\_MAIN}}$ – TX_LSB => Data Bus
$\overline{\text{ASSERT\_MSB\_MAIN}}$ – TX_MSB => Data Bus
ASSERT_ADDRESS – TX => Address Bus

Sheet: /
File: transfer-register.kicad_sch
Title:
Size: User Date: Rev:
KiCad E.D.A. kicad (6.0.0-0) Id: 1/1