Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

# FreeSoC 8051 Board User's Manual

This manual will help you get started using your FreeSoC as an 8051 emulator with neat new Special Function Registers. It will also detail the FreeSoC's use as a PSoC-based microcontroller for other projects.

**How can we use the FreeSoC?**

Mode 1: The FreeSoC is 40-pin DIP compatible microcontroller breakout with USB programming capability. The FreeSoC uses Cypress' powerful PSoC 5 chip. The PSoC, unlike other modern system-on-chips, contains not only a microcontroller and programmable and highly configurable digital blocks, but programmable analog blocks as well. This chip, along with its intuitive IDE PSoC Creator, makes changing designs and configuring hardware much easier for students and designers.

Mode 2: The board can be programmed (via USB from PSoC Creator) with an 8051 emulator project with special modifications. It can be used as an exact substitute for the Intel 8051 and can be programmed with 8051 assembly code from external memory or via USB using a batch file as detailed below.

**What's New?**

The FreeSoC 8051 emulators come in two flavors:

1. PWM: 8051 with 6 PWM generators (for power electronics applications)
2. SIG: 8051 with 2 ADCs, 2 DACs and 1 PWM generator (for signal processing applications)

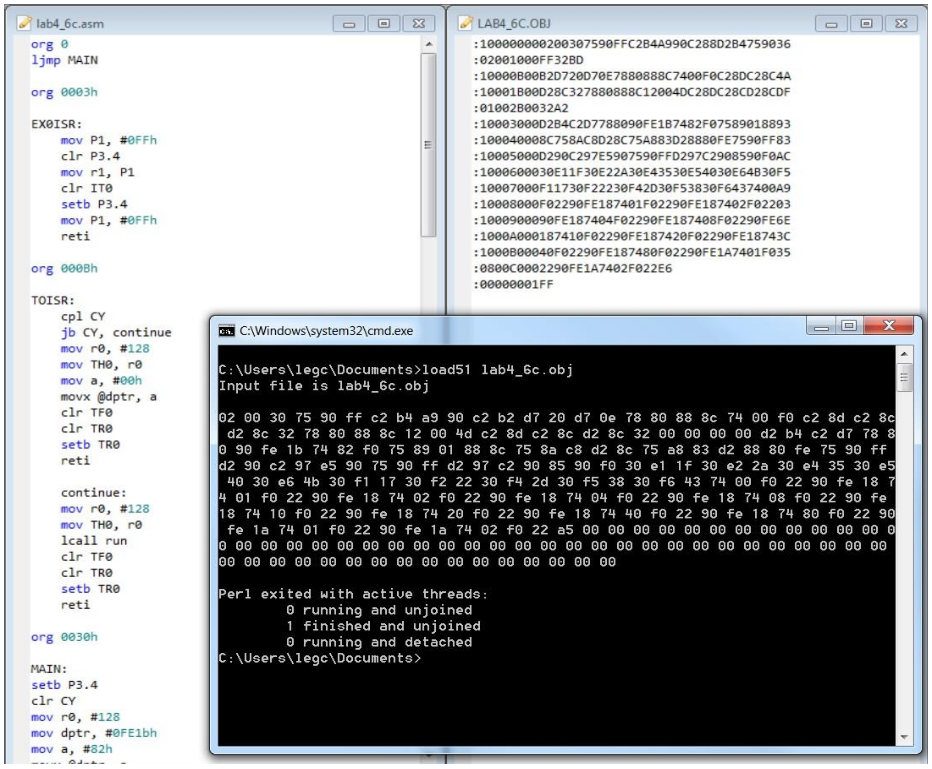
Pick the new functionalities you would like to add to your 8051 and program your FreeSoC with the associated project found on the course website. To do so, use PSoC Creator (see website for further documentation).

**Assembling your code**

You may use an 8051 assembler of your choice. The FreeSoC takes the generated Intel Hex (.hex) file.

**Loading your code onto the FreeSoC board**

Once you have assembled your PWM code, you will need to load it into the PSoC on-chip flash. To do this, run the batch file "load51" with your Intel Hex file. You will see the contents of your Intel Hex file print to your Window's terminal if successfully loaded.



lab4_6c.asm

```
org 0
ljmp MAIN

org 0003h

EX0ISR:
    mov P1, #0FFh
    clr P3.4
    mov r1, P1
    clr IT0
    setb P3.4
    mov P1, #0FFh
    reti

org 000Bh

TOISR:
    cpl CY
    jb CY, continue
    mov r0, #128
    mov TH0, r0
    mov a, #00h
    movx @dptr, a
    clr TF0
    clr TR0
    setb TR0
    reti

    continue:
    mov r0, #128
    mov TH0, r0
    lcall run
    clr TF0
    clr TR0
    setb TR0
    reti

org 0030h

MAIN:
setb P3.4
clr CY
mov r0, #128
mov dptr, #0FE1bh
mov a, #82h
```

LAB4_6C.OBJ

```
:1000000002003075 90FFC2B4A990C288D2B4759036
```

```
:100000000200307590FFC2B4A990C288D2B4759036
:02001000FF32BD
:10000B00B2D720D70E7880888C7400F0C28DC28C4A
:10001B00D28C327880888C12004DC28DC28CD28CDF
:01002B0032A2
:10003000D2B4C2D7788090FE1B7482F07589018893
:100040008C758AC8D28C75A883D28880FE7590FF83
:10005000D290C297E5907590FFD297C2908590F0AC
:1000600030E11F30E22A30E43530E54030E64B30F5
:10007000F11730F22230F42D30F53830F6437400A9
:10008000F02290FE187401F02290FE187402F02203
:1000900090FE187404F02290FE187408F02290FE6E
:1000A000187410F02290FE187420F02290FE18743C
:1000B00040F02290FE187480F02290FE1A7401F035
:0800C0002290FE1A7402F022E6
:00000001FF
```

C:\Windows\system32\cmd.exe

```
C:\Users\legc\Documents>load51 lab4_6c.obj
Input file is lab4_6c.obj

02 00 30 75 90 ff c2 b4 a9 90 c2 b2 d7 20 d7 0e 78 80 88 8c 74 00 f0 c2 8d c2 8c
 d2 8c 32 78 80 88 8c 12 00 4d c2 8d c2 8c d2 8c 32 00 00 00 00 d2 b4 c2 d7 78 8
0 90 fe 1b 74 82 f0 75 89 01 88 8c 75 8a c8 d2 8c 75 a8 83 d2 88 80 fe 75 90 ff
d2 90 c2 97 e5 90 75 90 ff d2 97 c2 90 85 90 f0 30 e1 1f 30 e2 2a 30 e4 35 30 e5
 40 30 e6 4b 30 f1 17 30 f2 22 30 f4 2d 30 f5 38 30 f6 43 74 00 f0 22 90 fe 18 7
4 01 f0 22 90 fe 18 74 02 f0 22 90 fe 18 74 04 f0 22 90 fe 18 74 08 f0 22 90 fe
18 74 10 f0 22 90 fe 18 74 20 f0 22 90 fe 18 74 40 f0 22 90 fe 18 74 80 f0 22 90
 fe 1a 74 01 f0 22 90 fe 1a 74 02 f0 22 a5 00 00 00 00 00 00 00 00 00 00 00 00 0
0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Perl exited with active threads:
        0 running and unjoined
        1 finished and unjoined
        0 running and detached
C:\Users\legc\Documents>
```

**Figure 1 Left: This is an .asm file. Right: The assembled Intel Hex file of the .asm file on the left. Bottom: Loading the code into FreeSoC flash with “load51.”**

## SECTION 1: PWM 8051 Emulator

### *Creating PWM waveforms*

1. Two complementary PWM waves with configurable period, duty cycle and delay

The PSoC uses configurable digital blocks to create a variety of digital functions. This emulated 8051 microcontroller uses a PSoC PWM generator in its special PWM configuration. We can write to specific registers in the PSoC to change the period, duty cycle and delays of our two 180° phase-shifted square waves.

The comments on the right detail the contents of each of these special registers.

```
;==============================================================
; PWM TEMPLATE
;
; This is an template for creating two opposite PWM waves with
; variable frequency, duty cycle and delay. It is written in
; 8051 assembly code for use on the enhanced 8051 FreeSoC
; designed for this class.
;==============================================================

;******************************************************************************
; EDIT THESE CONSTANTS
```

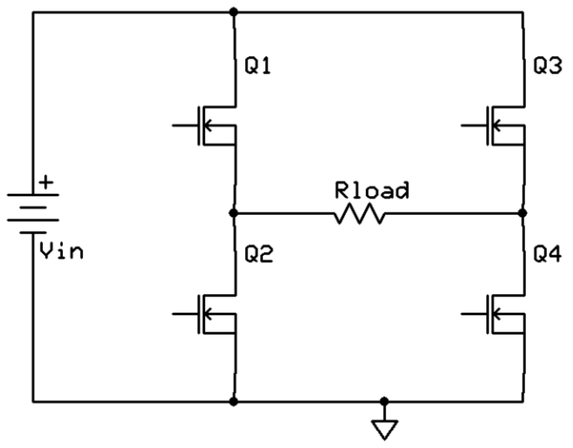
```
P equ 65535                 ; Set period. Max = 65535 (5.2ms, 190 Hz)
D equ 32222                 ; Set duty cycle period. For 50%, D = P/2
K equ 5                     ; Set # cycle of delay (2-256 cycles of 0.1us)
W equ 0                     ; Set time for phase difference delay (us)

;*******************************************************************************

ljmp MAIN
org 0030h
MAIN:
        mov dptr, #P        ; Store period in 16-bit register "dptr"
        mov 9Bh, dph        ; Store high byte in register 0x9B
        mov 9Ch, dpl        ; Store low byte in register 0x9C

        mov dptr, #D        ; Store duty cycle period in dptr
        mov 9Dh, dph        ; Store high byte in register 0x9D
        mov 9Eh, dpl        ; Store low byte in register 0x9E

        mov 9Fh, #K         ; Store delay in register 0x9F

        mov dptr, #W        ; Store phase difference in dptr
        mov 0A2h, dph       ; Store high byte in register 0xA2
        mov 0A3h, dpl       ; Store low byte in register 0xA3

        setb 0C0h           ; This enables the PWM source

        loop: sjmp loop
```

**Figure 2 This template may be changed to create the delay and inverted delay signals as inputs to the FET drivers of a totem circuit.**

We can use P, D and K to recreate the DELAY and $\overline{\text{DELAY}}$ we have been using a combination of ICs to create! Just set the frequency, duty cycle and delay amounts that you need.

2. H-bridge waveforms with adjustable phase difference

Now we want to make an inverter to convert some DC voltage to an AC waveform. We might want to make one with the control waveforms offset by 180°, or by some other phase shift.



We can add the W parameter to introduce a phase delay in microseconds for the second and third PWM sources. The phase delay between the first and second and second and third sources will be the same.

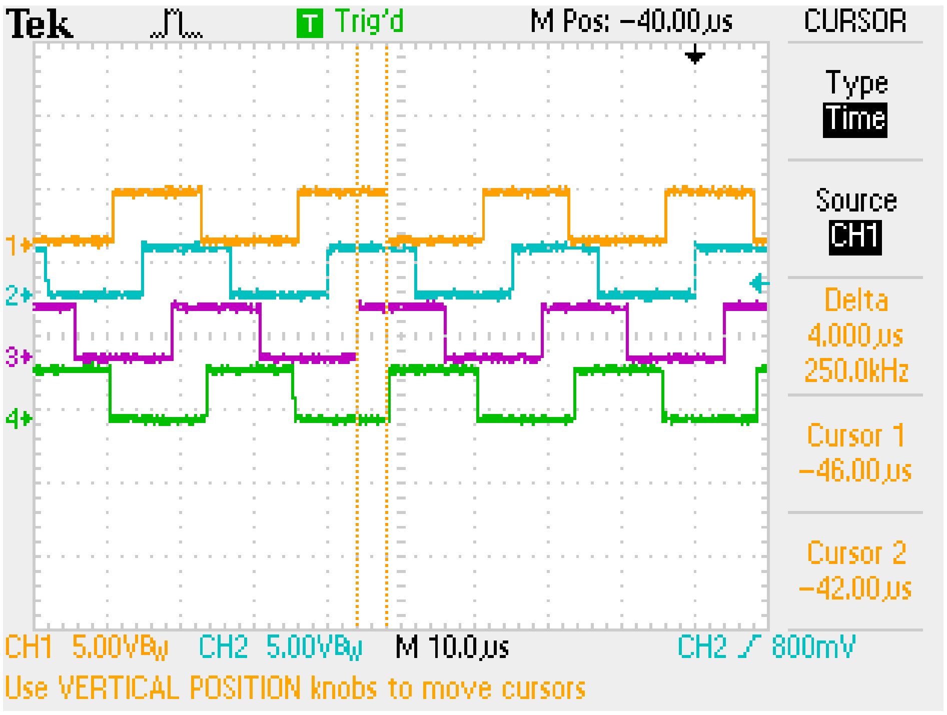
3. 3-phase inverter waveforms with adjustable phase difference

We can also make a 3-phase inverter for induction machine drives. PWM waveforms with phase delays of 180° and 120° or arbitrary phase delays can be constructed using the above form.

Using the example from the class handout, we can create these waveforms with the parameters below:

frequency = 40kHz
duty cycle = 0.5
deadtime = 0ns
phase delay = 120°

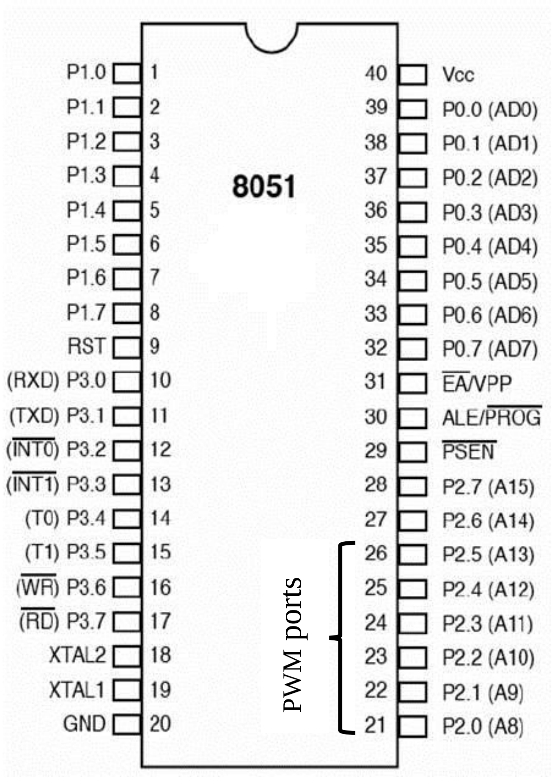
```
;*************************************************************************************
; EDIT THESE CONSTANTS

P equ 250                        ; Set period. Max = 65535 (5.2ms, 190 Hz)
D equ 125                        ; Set duty cycle period. For 50%, D = P/2
K equ 5                          ; Set # cycle of delay (2-256 cycles of 0.1us)
W equ 4                          ; Set time for phase difference delay (us)

;*************************************************************************************
```



*Using the board*

Below is a pinout of the standard 8051 microcontroller. Also shown is the standard memory map of the 8051 with extended PWM FreeSoC memory map (Tables 1 and 2).

The FreeSoC firmware is hardcoded to use Port 2 as its PWM output port. Pins P2.0 – P2.5 will output your (up to) 3 independent PWM waves with adjustable deadtime, with P2.0, P2.2 and P2.4 being your first, second and third sources respectively and P2.1, P2.3 and P2.5 being their complements.



This document is meant to serve as a supplement to the Intel MCS® 51 Microcontroller Family User's Manual. The FreeSoC was designed to be an 8051 emulator with almost all of Intel's original specifications. This document details the operation of the new PWM Special Function Registers, but should be used in conjunction with the Intel manual if true 8051 functionality is desired.

Once your FreeSoC code has been assembled and you have loaded it onto the board via the USB, you may either use the 3.3V from the USB to continue to power the board, or power it externally with 5V on $V_{CC}$.

WARNING: DO NOT POWER THE BOARD WITH MORE THAN +5V. These boards are expensive and cannot handle a supply voltage higher than 5V. See Appendix A for a schematic of the FreeSoC board.

*FreeSoC PWM Memory Map*

**Table 1 PWM modified SFR space for FreeSoC with added SFRs in bold**

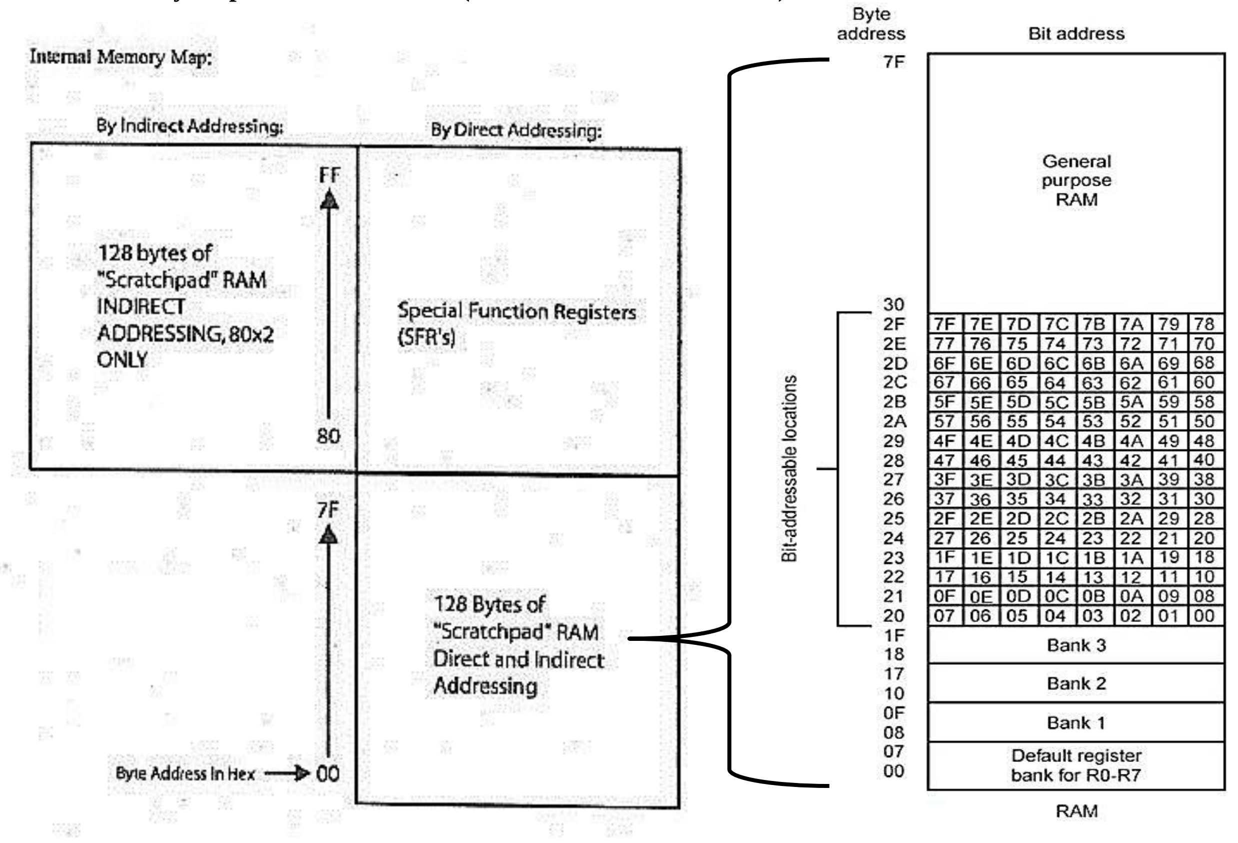
| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| F8h | | | | | | | | | FFh |
| F0h | B | | | | | | | | F7h |
| E8h | | | | | | | | | EFh |
| E0h | ACC | | | | | | | | E7h |
| D8h | | | | | | | | | DFh |
| D0h | PSW | | | | | | | | D7h |
| C8h | | | | | | | | | CFh |
| C0h | **ENP** | | | | | | | | C7h |
| B8h | IP | | | | | | | | BFh |
| B0h | P3 | | | | | | | | B7h |
| A8h | IE | | | | | | | | AFh |
| A0h | P2 | | | | | | | | A7h |
| 98h | SCON | SBUF | | | | | | | 9Fh |
| 90h | P1 | **PWMPH** | **PWMPL** | **PWMFH** | **PWMFL** | **PWMDH** | **PWMDL** | **PWMK** | 97h |
| 88h | TCON | TMOD | TL0 | TL1 | TH0 | TH1 | | | 8Fh |
| 80h | P0 | SP | DPL | DPH | | | | PCON | 87h |

PWMPH, PWMPL are high and low bytes of PWM phase delay register
PWMDH, PWMDL are high and low bytes of PWM duty cycle register
PWMK is PWM dead-time register
PWMFH, PWMFL are high and low bytes of PWM period (frequency) register

## Differences between Intel 8051 and FreeSoC

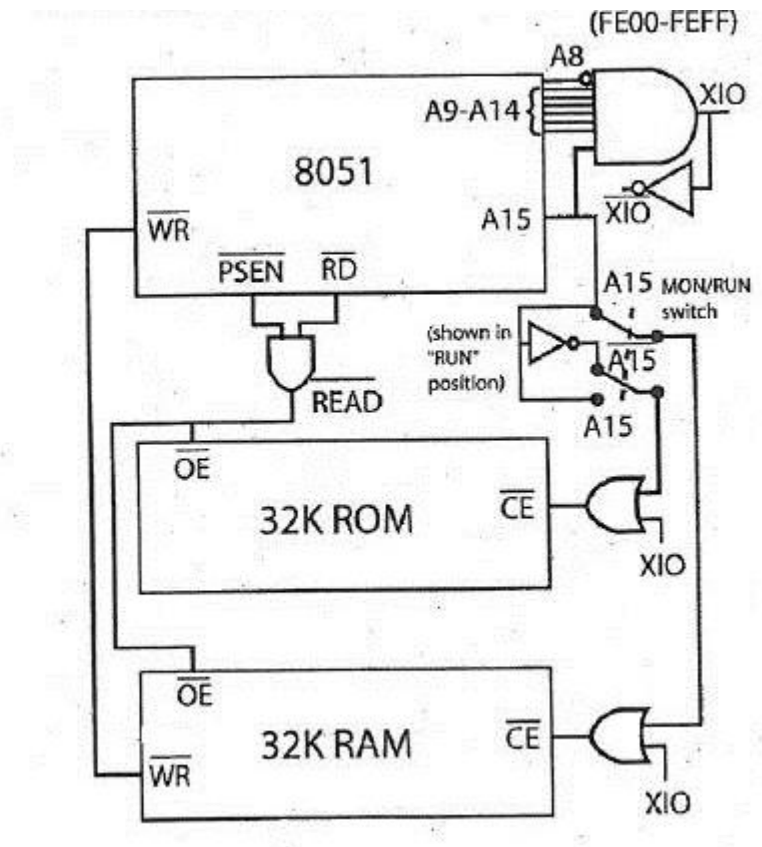
- 8052-specific hardware (Timer 2, etc) is not implemented.
- The FreeSoC can run much faster when it is not constantly reading program code from external ROM. Therefore, if external ROM is used (as in the R31JP) the FreeSoC will read the contents of this ROM into its 32K reserved 8051 program space.
    - When the R31JP is used, the FreeSoC will grab the 32K of code in whichever external memory is at address 0x0000.
- If external RAM is not used, the FreeSoC will use its 8K of reserved 8051 RAM space to be accessed with MOVX commands.
- Timer 0 and Timer 1 modes 0 (13-bit mode) and mode 3 (Timer 0: TL0 and TH0 8-bit counter mode) are not implemented.
- P3.3 (INT1) and P3.5 (T1) have been taken for use as the ADC input and DAC output respectively.

**Table 2 Memory map for standard 8051 (taken from 6.115 Lecture 1)**


Internal Memory Map:
By Indirect Addressing:
By Direct Addressing:
FF
128 bytes of "Scratchpad" RAM INDIRECT ADDRESSING, 80x2 ONLY
80
Special Function Registers (SFR's)
7F
128 Bytes of "Scratchpad" RAM Direct and Indirect Addressing
Byte Address In Hex
00
Byte address
Bit address
7F
General purpose RAM
30
Bit-addressable locations
1F
18
Bank 3
17
10
Bank 2
0F
08
Bank 1
07
00
Default register bank for R0-R7
RAM

| Byte address | Bit address | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 2F | 7F | 7E | 7D | 7C | 7B | 7A | 79 | 78 |
| 2E | 77 | 76 | 75 | 74 | 73 | 72 | 71 | 70 |
| 2D | 6F | 6E | 6D | 6C | 6B | 6A | 69 | 68 |
| 2C | 67 | 66 | 65 | 64 | 63 | 62 | 61 | 60 |
| 2B | 5F | 5E | 5D | 5C | 5B | 5A | 59 | 58 |
| 2A | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 |
| 29 | 4F | 4E | 4D | 4C | 4B | 4A | 49 | 48 |
| 28 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 |
| 27 | 3F | 3E | 3D | 3C | 3B | 3A | 39 | 38 |
| 26 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 |
| 25 | 2F | 2E | 2D | 2C | 2B | 2A | 29 | 28 |
| 24 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 |
| 23 | 1F | 1E | 1D | 1C | 1B | 1A | 19 | 18 |
| 22 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |
| 21 | 0F | 0E | 0D | 0C | 0B | 0A | 09 | 08 |
| 20 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |

## More on the R31JP

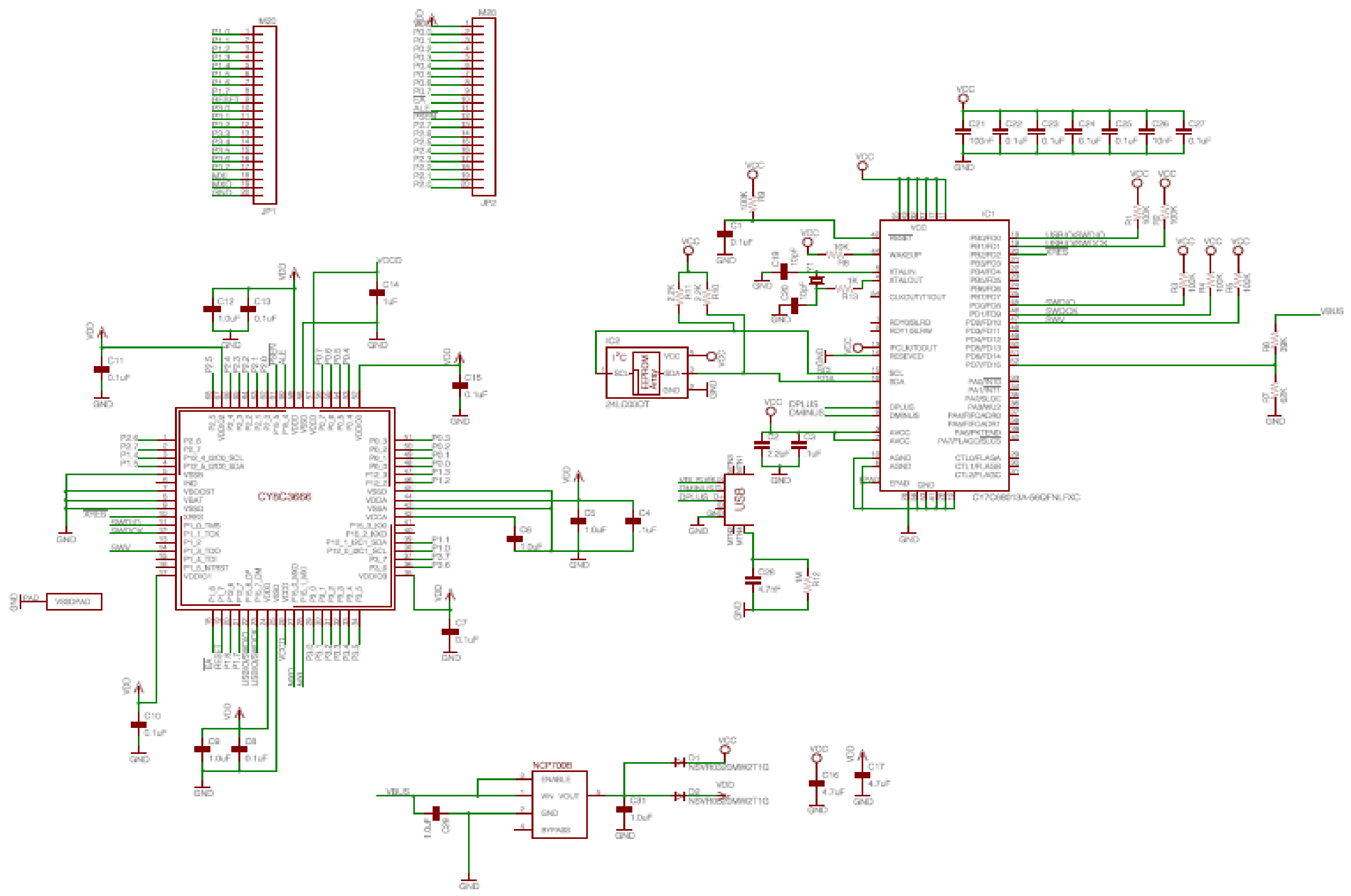


In normal R31JP operation either the external ROM or RAM, accessed by PSEN and RD control signals respectively, can be read from using the combined READ signal (OR for active low signals).

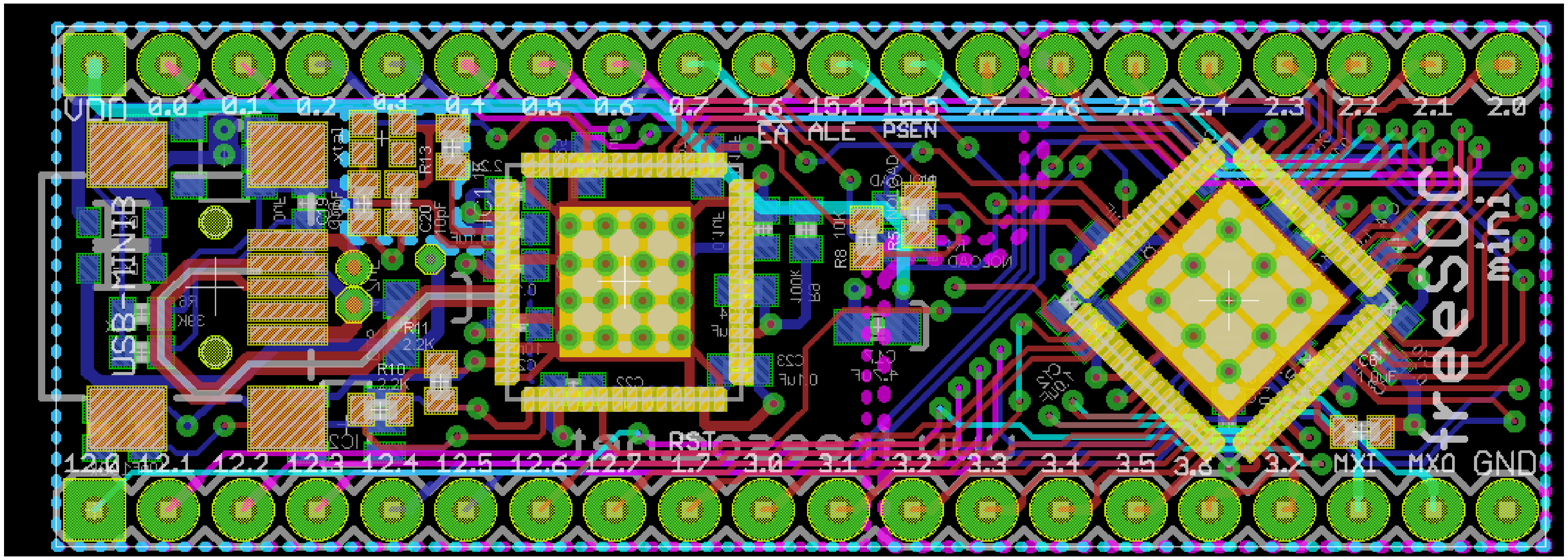
Addresses in the 0x0000-0x7FFF range will access the external RAM in "RUN" mode and ROM in "MON" mode. Using the 0xFE00-0xFEFF range will select memory-mapped IO devices via XIO.

**Figure 3 R31JP operation**

**Appendix A.** Board Schematic



**Appendix B.** Board Layout and Photo


VDD
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
1.6
EA
15.1
ALE
15.5
PSEN
2.7
2.6
2.5
2.4
2.3
2.2
2.1
2.0
USB-MINIB
xtal
R13
R11
2.2K
R10
R8 10K
RST
12.0
12.1
12.2
12.3
12.4
12.5
12.6
12.7
1.7
3.0
3.1
3.2
3.3
3.4
3.5
3.6
3.7
MXI
MXO
GND
freeSoC
mini