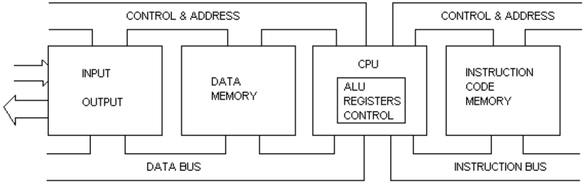
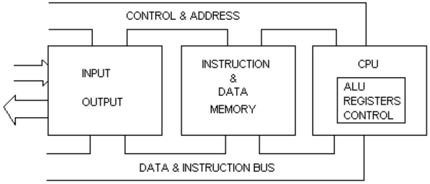
dsPIC33 controllers with MPLAB C30 compiler

Šimon Řeřucha
(res@isibrno.cz)

Harvard vs. von Neumann

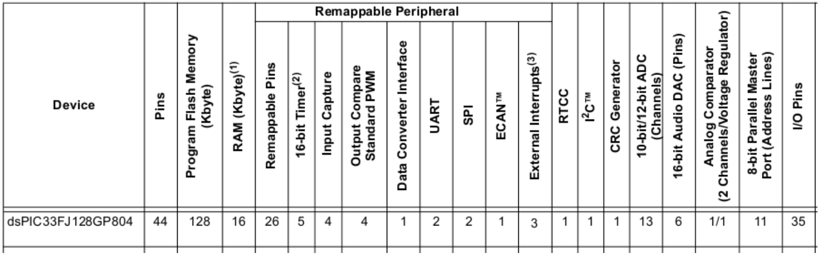


HARVARD ARCHITECTURE MICROPROCESSOR



VON NEUMANN ARCHITECTURE MICROPROCESSOR

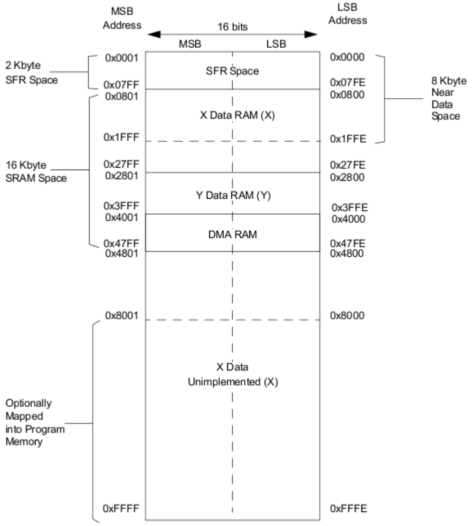
Family

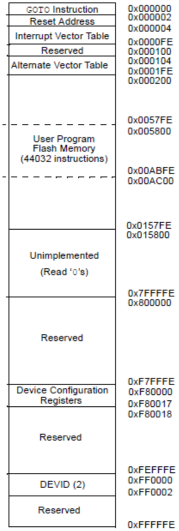


Device	Pins	Program Flash Memory (Kbyte)	RAM (Kbyte)[1]	Remappable Peripheral									RTCC	I2C™	CRC Generator	10-bit/12-bit ADC (Channels)	16-bit Audio DAC (Pins)	Analog Comparator (2 Channels/Voltage Regulator)	8-bit Parallel Master Port (Address Lines)	I/O Pins
				Remappable Pins	16-bit Timer[2]	Input Capture	Output Compare Standard PWM	Data Converter Interface	UART	SPI	ECAN™	External Interrupts[3]								
dsPIC33FJ128GP804	44	128	16	26	5	4	4	1	2	2	1	3	1	1	1	13	6	1/1	11	35

- 16-bit Digital Signal Controller
- Modified-Harvard architecture
- 24-bit address

Memory Map


MSB Address
16 bits
LSB Address
MSB
LSB
2 Kbyte SFR Space
0x0001
0x07FF
0x0801
SFR Space
0x0000
0x07FE
0x0800
8 Kbyte Near Data Space
X Data RAM (X)
0x1FFF
0x1FFE
16 Kbyte SRAM Space
0x27FF
0x2801
0x27FE
0x2800
Y Data RAM (Y)
0x3FFF
0x4001
0x3FFE
0x4000
DMA RAM
0x47FF
0x4801
0x47FE
0x4800
0x8001
0x8000
X Data Unimplemented (X)
Optionally Mapped into Program Memory
0xFFFF
0xFFFE


GOTO Instruction
Reset Address
Interrupt Vector Table
Reserved
Alternate Vector Table
User Program Flash Memory (44032 instructions)
Unimplemented (Read '0's)
Reserved
Device Configuration Registers
Reserved
DEVID (2)
Reserved
0x000000
0x000002
0x000004
0x0000FE
0x000100
0x000104
0x0001FE
0x000200
0x0057FE
0x005800
0x00ABFE
0x00AC00
0x0157FE
0x015800
0x7FFFFE
0x800000
0xF7FFFE
0xF80000
0xF80017
0xF80018
0xFEFFFE
0xFF0000
0xFF0002
0xFFFFFE

Compiler: Accessing SFR

Special Function Registers Example:

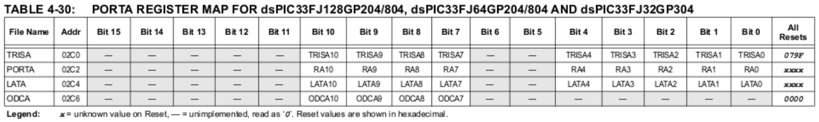


TABLE 4-30: PORTA REGISTER MAP FOR dsPIC33FJ128GP204/804, dsPIC33FJ64GP204/804 AND dsPIC33FJ32GP304

File Name	Addr	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	All Resets
TRISA	02C0	—	—	—	—	—	TRISA10	TRISA9	TRISA8	TRISA7	—	—	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	*079F*
PORTA	02C2	—	—	—	—	—	RA10	RA9	RA8	RA7	—	—	RA4	RA3	RA2	RA1	RA0	*xxxx*
LATA	02C4	—	—	—	—	—	LATA10	LATA9	LATA8	LATA7	—	—	LATA4	LATA3	LATA2	LATA1	LATA0	*xxxx*
ODCA	02C6	—	—	—	—	—	ODCA10	ODCA9	ODCA8	ODCA7	—	—	—	—	—	—	—	*0000*

Legend: *x* = unknown value on Reset, — = unimplemented, read as '0'. Reset values are shown in hexadecimal.

```
// set all to outputs
TRISA = 0x0000;

// set RA4 to input
TRISAbits.TRISA4 = 1;
```

Configuration Bits (fuses)

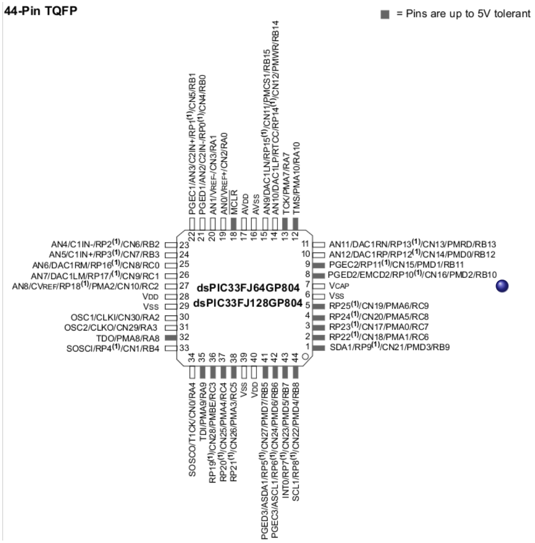
- small block of non-volatile memory
- basic initial configuration, e.g.
- security setting
- oscillator setting
- watchdog, ...

For description, see datasheet Chapter 28.1

Ex. Fuses

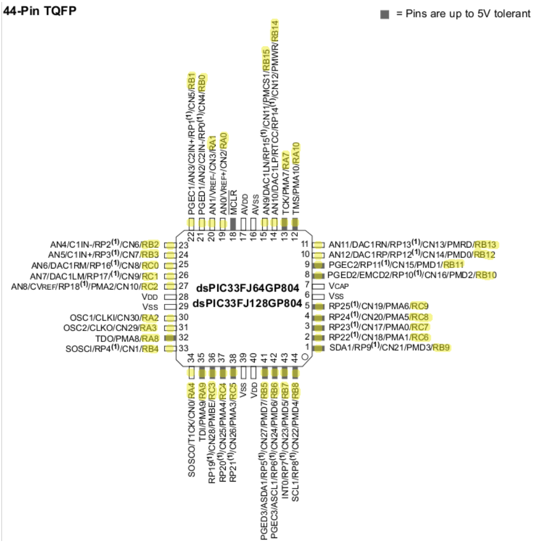
```
//External XTAL
_FOSCSEL ( FNOSC_PRI );

// HS mode, 10-40MHZ + IO mapping lock
_FOSC( POSCMD_HS & IOL1WAY_OFF );

// turn off watchdog
_FWDT(FWDTEN_OFF);
```

μC Pinout



For description, see datasheet p. 17-19.

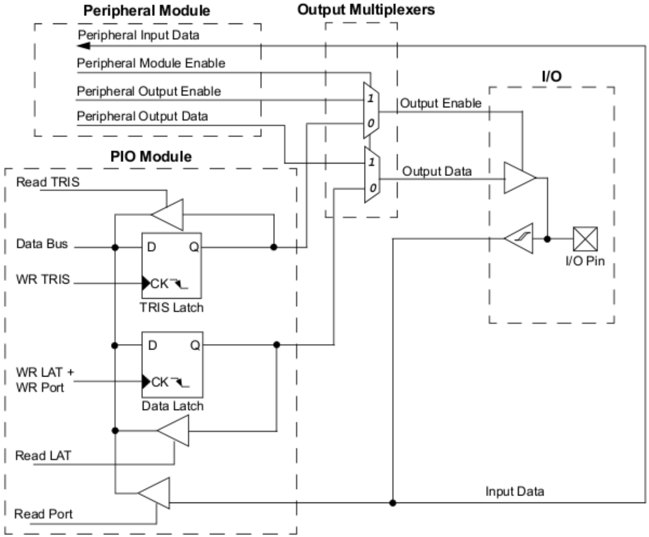
Parallel I/O


44-Pin TQFP
■ = Pins are up to 5V tolerant
dsPIC33FJ64GP804
dsPIC33FJ128GP804
PGEC1/AN3/C2IN+/RP1(1)/CN5/RB1
PGED1/AN2/C2IN-/RP0(1)/CN4/RB0
AN1/VREF-/CN3/RA1
AN0/VREF+/CN2/RA0
MCLR
AVDD
AVSS
AN9/DAC1LN/RP15(1)/CN11/PMCS1/RB15
AN10/DAC1LP/RTCC/RP14(1)/CN12/PMWR/RB14
TCK/PMA7/RA7
TMS/PMA10/RA10
AN4/C1IN-/RP2(1)/CN6/RB2
AN5/C1IN+/RP3(1)/CN7/RB3
AN6/DAC1RM/RP16(1)/CN8/RC0
AN7/DAC1LM/RP17(1)/CN9/RC1
AN8/CVREF/RP18(1)/PMA2/CN10/RC2
VDD
VSS
OSC1/CLKI/CN30/RA2
OSC2/CLKO/CN29/RA3
TDO/PMA8/RA8
SOSCI/RP4(1)/CN1/RB4
AN11/DAC1RN/RP13(1)/CN13/PMRD/RB13
AN12/DAC1RP/RP12(1)/CN14/PMD0/RB12
PGEC2/RP11(1)/CN15/PMD1/RB11
PGED2/EMCD2/RP10(1)/CN16/PMD2/RB10
VCAP
VSS
RP25(1)/CN19/PMA6/RC9
RP24(1)/CN20/PMA5/RC8
RP23(1)/CN17/PMA0/RC7
RP22(1)/CN18/PMA1/RC6
SDA1/RP9(1)/CN21/PMD3/RB9
SOSCO/T1CK/CN0/RA4
TDI/PMA9/RA9
RP19(1)/CN28/PMBE/RC3
RP20(1)/CN25/PMA4/RC4
RP21(1)/CN26/PMA3/RC5
VSS
VDD
PGED3/ASDA1/RP5(1)/CN27/PMD7/RB5
PGEC3/ASCL1/RP6(1)/CN24/PMD6/RB6
INT0/RP7(1)/CN23/PMD5/RB7
SCL1/RP8(1)/CN22/PMD4/RB8

Parallel I/O

- general purpose digital I/O
- I/O pins shared with peripherals
- bidirectional ports, output latched
- Schmitt-triggered
- optionally open collector

Parallel I/O


Peripheral Module
Output Multiplexers
Peripheral Input Data
Peripheral Module Enable
Peripheral Output Enable
Peripheral Output Data
I/O
Output Enable
Output Data
1
0
1
0
PIO Module
Read TRIS
Data Bus
WR TRIS
D
Q
CK
TRIS Latch
D
Q
CK
Data Latch
WR LAT +
WR Port
Read LAT
Read Port
I/O Pin
Input Data

PIO registers

All I/O ports have four registers directly associated with the operation of the port, where 'x' is a letter that denotes the particular I/O port:

- TRISx: Data Direction registers
- PORTx: I/O Port registers
- LATx: I/O Latch registers
- ODCx: Open-Drain Control registers

Each I/O pin on the device has an associated bit in the TRISx, PORTx, and LATx registers.

The differences between the PORTx and LATx registers can be summarized as follows:

- A write to the PORTx register writes the data value to the port latch
- A write to the LATx register writes the data value to the port latch
- A read of the PORTx register reads the data value on the I/O pin
- A read of the LATx register reads the data value held in the port latch

GPIO on dsPIC33-FJ128-MC804

- three (incomplete) 16-bit ports A[10:7,4:0], B[15:0], C[9:0]
- Peripheral Pin Select technique – arbitrary remapping of peripheral I/O
- optional pull-ups
- optional Change Notification (generates interrupt on state change)
- note the analog / digital multiplexing (register `AD1PCFG`)

I/O Port A,B,C

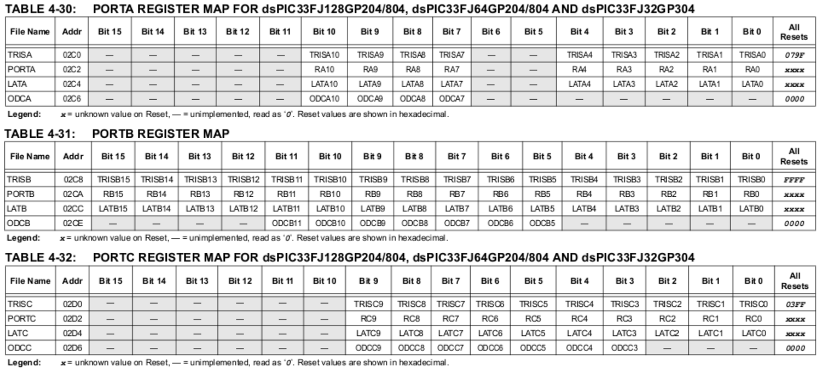


TABLE 4-30: PORTA REGISTER MAP FOR dsPIC33FJ128GP204/804, dsPIC33FJ64GP204/804 AND dsPIC33FJ32GP304

File Name	Addr	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	All Resets
TRISA	02C0	—	—	—	—	—	TRISA10	TRISA9	TRISA8	TRISA7	—	—	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	*079F*
PORTA	02C2	—	—	—	—	—	RA10	RA9	RA8	RA7	—	—	RA4	RA3	RA2	RA1	RA0	*xxxx*
LATA	02C4	—	—	—	—	—	LATA10	LATA9	LATA8	LATA7	—	—	LATA4	LATA3	LATA2	LATA1	LATA0	*xxxx*
ODCA	02C6	—	—	—	—	—	ODCA10	ODCA9	ODCA8	ODCA7	—	—	—	—	—	—	—	*0000*

Legend: *x* = unknown value on Reset, — = unimplemented, read as '*0*'. Reset values are shown in hexadecimal.

TABLE 4-31: PORTB REGISTER MAP

File Name	Addr	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	All Resets
TRISB	02C8	TRISB15	TRISB14	TRISB13	TRISB12	TRISB11	TRISB10	TRISB9	TRISB8	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	*FFFF*
PORTB	02CA	RB15	RB14	RB13	RB12	RB11	RB10	RB9	RB8	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	*xxxx*
LATB	02CC	LATB15	LATB14	LATB13	LATB12	LATB11	LATB10	LATB9	LATB8	LATB7	LATB6	LATB5	LATB4	LATB3	LATB2	LATB1	LATB0	*xxxx*
ODCB	02CE	—	—	—	—	ODCB11	ODCB10	ODCB9	ODCB8	ODCB7	ODCB6	ODCB5	—	—	—	—	—	*0000*

Legend: *x* = unknown value on Reset, — = unimplemented, read as '*0*'. Reset values are shown in hexadecimal.

TABLE 4-32: PORTC REGISTER MAP FOR dsPIC33FJ128GP204/804, dsPIC33FJ64GP204/804 AND dsPIC33FJ32GP304

File Name	Addr	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	All Resets
TRISC	02D0	—	—	—	—	—	—	TRISC9	TRISC8	TRISC7	TRISC6	TRISC5	TRISC4	TRISC3	TRISC2	TRISC1	TRISC0	*03FF*
PORTC	02D2	—	—	—	—	—	—	RC9	RC8	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	*xxxx*
LATC	02D4	—	—	—	—	—	—	LATC9	LATC8	LATC7	LATC6	LATC5	LATC4	LATC3	LATC2	LATC1	LATC0	*xxxx*
ODCC	02D6	—	—	—	—	—	—	ODCC9	ODCC8	ODCC7	ODCC6	ODCC5	ODCC4	ODCC3	—	—	—	*0000*

Legend: *x* = unknown value on Reset, — = unimplemented, read as '*0*'. Reset values are shown in hexadecimal.

Using GPIO

- locate your ports (Board / Kit documentation)
- set ports to digital (`AD1PCFG`)
  `ANx` are analog by default!
- set correct direction (`TRISA`, `TRISB`, `TRISC`)
  $0 =$ output, $1 =$ input
- read from `PORTA`, `PORTB`, `PORTC`
- write to `PORTA`, `PORTB`, `PORTC` or `LATA`, `LATB`, `LATC`

Timers on dsPIC33-FJ128-MC804

- 5x 16-bit timers/counters
- Timer1: time counter for the real-time clock)
- Timer2+3 and 4+5 chaining for 32-bit counters

For description, see datasheet Chapters 12, 13 + Family Reference Manual, Chapter 11 (Timers)

TMR Registers

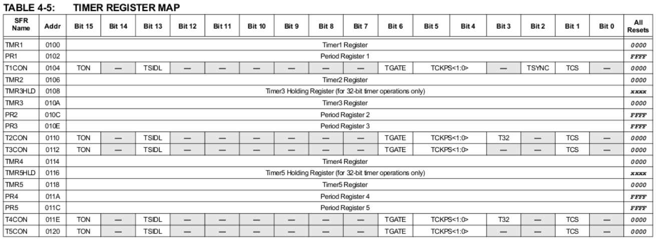


TABLE 4-5: TIMER REGISTER MAP

SFR Name	Addr	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	All Resets
TMR1	0100	Timer1 Register																*0000*
PR1	0102	Period Register 1																*FFFF*
T1CON	0104	TON	—	TSIDL	—	—	—	—	—	—	TGATE	TCKPS<1:0>		—	TSYNC	TCS	—	*0000*
TMR2	0106	Timer2 Register																*0000*
TMR3HLD	0108	Timer3 Holding Register (for 32-bit timer operations only)																*xxxx*
TMR3	010A	Timer3 Register																*0000*
PR2	010C	Period Register 2																*FFFF*
PR3	010E	Period Register 3																*FFFF*
T2CON	0110	TON	—	TSIDL	—	—	—	—	—	—	TGATE	TCKPS<1:0>		T32	—	TCS	—	*0000*
T3CON	0112	TON	—	TSIDL	—	—	—	—	—	—	TGATE	TCKPS<1:0>		—	—	TCS	—	*0000*
TMR4	0114	Timer4 Register																*0000*
TMR5HLD	0116	Timer5 Holding Register (for 32-bit timer operations only)																*xxxx*
TMR5	0118	Timer5 Register																*0000*
PR4	011A	Period Register 4																*FFFF*
PR5	011C	Period Register 5																*FFFF*
T4CON	011E	TON	—	TSIDL	—	—	—	—	—	—	TGATE	TCKPS<1:0>		T32	—	TCS	—	*0000*
T5CON	0120	TON	—	TSIDL	—	—	—	—	—	—	TGATE	TCKPS<1:0>		—	—	TCS	—	*0000*

Interrupts on dsPIC33-FJ128-MC804

- 15 priority levels
  Up to eight processor exceptions and software traps

  Seven user-selectable priority levels
- Interrupt Vector Table (IVT) with up to 126 vectors
- A unique vector for each interrupt or exception source
- Fixed priority within a specified user priority level
- Fixed interrupt entry and return latencies

For description, see datasheet Family Reference Manual, Chapter 6 (Interrupts)

Ex. Change Notification Interrupts

```
// button 1 on B4 / CN1
TRISBbits.TRISB4 = 1;
CNPU1bits.CN1PUE = 0;
CNEN1bits.CN1IE = 1;

//define interrupt handler
...
```

Simple Interrupts

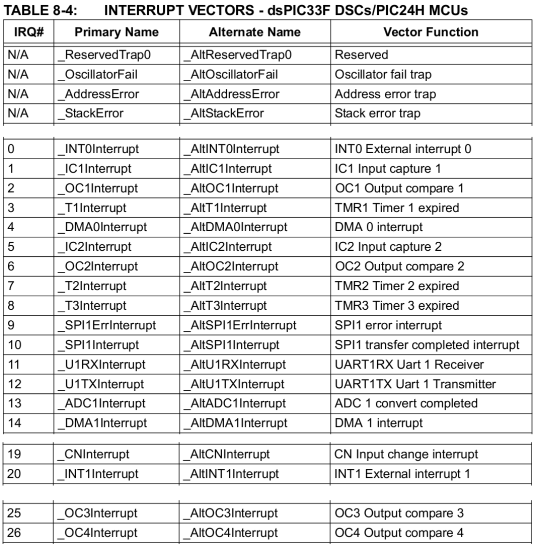


TABLE 8-4: INTERRUPT VECTORS - dsPIC33F DSCs/PIC24H MCUs

IRQ#	Primary Name	Alternate Name	Vector Function
N/A	_ReservedTrap0	_AltReservedTrap0	Reserved
N/A	_OscillatorFail	_AltOscillatorFail	Oscillator fail trap
N/A	_AddressError	_AltAddressError	Address error trap
N/A	_StackError	_AltStackError	Stack error trap
0	_INT0Interrupt	_AltINT0Interrupt	INT0 External interrupt 0
1	_IC1Interrupt	_AltIC1Interrupt	IC1 Input capture 1
2	_OC1Interrupt	_AltOC1Interrupt	OC1 Output compare 1
3	_T1Interrupt	_AltT1Interrupt	TMR1 Timer 1 expired
4	_DMA0Interrupt	_AltDMA0Interrupt	DMA 0 interrupt
5	_IC2Interrupt	_AltIC2Interrupt	IC2 Input capture 2
6	_OC2Interrupt	_AltOC2Interrupt	OC2 Output compare 2
7	_T2Interrupt	_AltT2Interrupt	TMR2 Timer 2 expired
8	_T3Interrupt	_AltT3Interrupt	TMR3 Timer 3 expired
9	_SPI1ErrInterrupt	_AltSPI1ErrInterrupt	SPI1 error interrupt
10	_SPI1Interrupt	_AltSPI1Interrupt	SPI1 transfer completed interrupt
11	_U1RXInterrupt	_AltU1RXInterrupt	UART1RX Uart 1 Receiver
12	_U1TXInterrupt	_AltU1TXInterrupt	UART1TX Uart 1 Transmitter
13	_ADC1Interrupt	_AltADC1Interrupt	ADC 1 convert completed
14	_DMA1Interrupt	_AltDMA1Interrupt	DMA 1 interrupt
19	_CNInterrupt	_AltCNInterrupt	CN Input change interrupt
20	_INT1Interrupt	_AltINT1Interrupt	INT1 External interrupt 1
25	_OC3Interrupt	_AltOC3Interrupt	OC3 Output compare 3
26	_OC4Interrupt	_AltOC4Interrupt	OC4 Output compare 4

Ex. Change Notification Interrupts

Technique for generating interrupts on port change (see Family Reference Manual, Chapter 10).

```
//define interrupt handler
void __attribute__((interrupt, no_auto_psv)) _CNInterrupt (void)
{

// state evaluation
if(PORTBbits.RB4 == 0) ... mach etwas ...

// Clear interrupt flag
IFS1bits.CNIF = 0;

return;
}
```

Port Change Interrupts

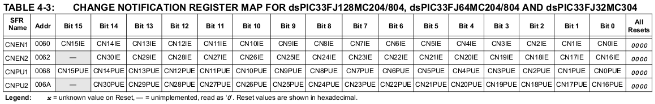
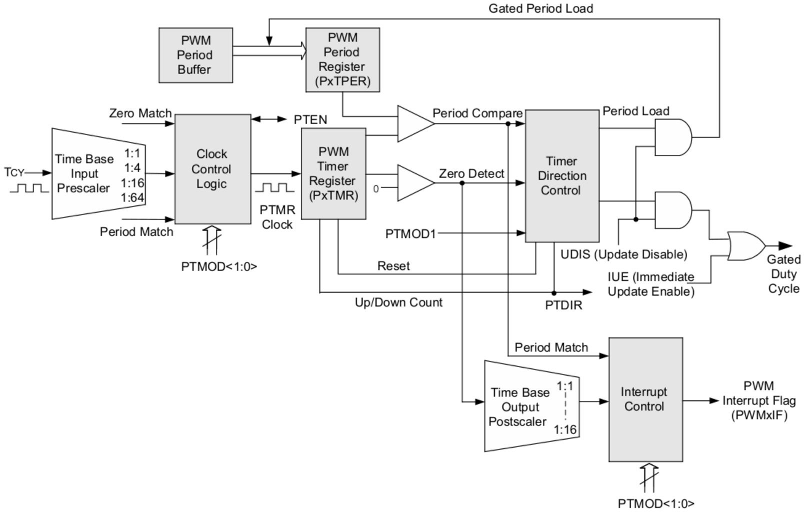


TABLE 4-3: CHANGE NOTIFICATION REGISTER MAP FOR dsPIC33FJ128MC204/804, dsPIC33FJ64MC204/804 AND dsPIC33FJ32MC304

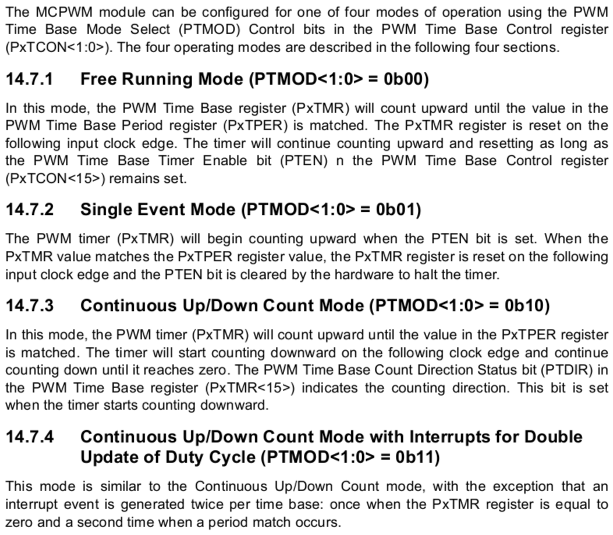
SFR Name	Addr	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	All Resets
CNEN1	0060	CN15IE	CN14IE	CN13IE	CN12IE	CN11IE	CN10IE	CN9IE	CN8IE	CN7IE	CN6IE	CN5IE	CN4IE	CN3IE	CN2IE	CN1IE	CN0IE	*0000*
CNEN2	0062	—	CN30IE	CN29IE	CN28IE	CN27IE	CN26IE	CN25IE	CN24IE	CN23IE	CN22IE	CN21IE	CN20IE	CN19IE	CN18IE	CN17IE	CN16IE	*0000*
CNPU1	0068	CN15PUE	CN14PUE	CN13PUE	CN12PUE	CN11PUE	CN10PUE	CN9PUE	CN8PUE	CN7PUE	CN6PUE	CN5PUE	CN4PUE	CN3PUE	CN2PUE	CN1PUE	CN0PUE	*0000*
CNPU2	006A	—	CN30PUE	CN29PUE	CN28PUE	CN27PUE	CN26PUE	CN25PUE	CN24PUE	CN23PUE	CN22PUE	CN21PUE	CN20PUE	CN19PUE	CN18PUE	CN17PUE	CN16PUE	*0000*

Legend: *x* = unknown value on Reset, — = unimplemented, read as '*0*'. Reset values are shown in hexadecimal.

Pulse Width Modulation



PWM modes


The MCPWM module can be configured for one of four modes of operation using the PWM Time Base Mode Select (PTMOD) Control bits in the PWM Time Base Control register (PxTCON<1:0>). The four operating modes are described in the following four sections.

14.7.1 Free Running Mode (PTMOD<1:0> = 0b00)

In this mode, the PWM Time Base register (PxTMR) will count upward until the value in the PWM Time Base Period register (PxTPER) is matched. The PxTMR register is reset on the following input clock edge. The timer will continue counting upward and resetting as long as the PWM Time Base Timer Enable bit (PTEN) n the PWM Time Base Control register (PxTCON<15>) remains set.

14.7.2 Single Event Mode (PTMOD<1:0> = 0b01)

The PWM timer (PxTMR) will begin counting upward when the PTEN bit is set. When the PxTMR value matches the PxTPER register value, the PxTMR register is reset on the following input clock edge and the PTEN bit is cleared by the hardware to halt the timer.

14.7.3 Continuous Up/Down Count Mode (PTMOD<1:0> = 0b10)

In this mode, the PWM timer (PxTMR) will count upward until the value in the PxTPER register is matched. The timer will start counting downward on the following clock edge and continue counting down until it reaches zero. The PWM Time Base Count Direction Status bit (PTDIR) in the PWM Time Base register (PxTMR<15>) indicates the counting direction. This bit is set when the timer starts counting downward.

14.7.4 Continuous Up/Down Count Mode with Interrupts for Double Update of Duty Cycle (PTMOD<1:0> = 0b11)

This mode is similar to the Continuous Up/Down Count mode, with the exception that an interrupt event is generated twice per time base: once when the PxTMR register is equal to zero and a second time when a period match occurs.

PWM registers

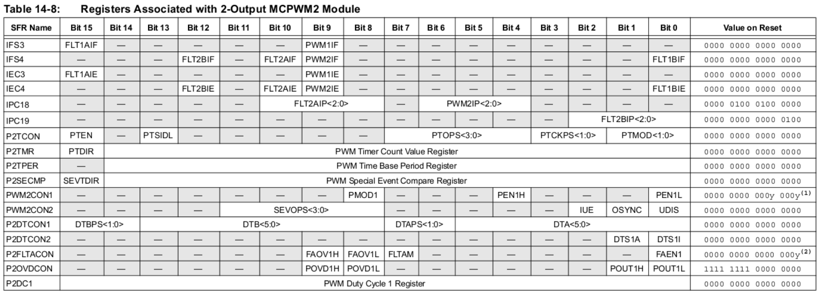


Table 14-8: Registers Associated with 2-Output MCPWM2 Module

SFR Name	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Reset
IFS3	FLT1AIF	—	—	—	—	—	PWM1IF	—	—	—	—	—	—	—	—	—	0000 0000 0000 0000
IFS4	—	—	—	FLT2BIF	—	FLT2AIF	PWM2IF	—	—	—	—	—	—	—	—	FLT1BIF	0000 0000 0000 0000
IEC3	FLT1AIE	—	—	—	—	—	PWM1IE	—	—	—	—	—	—	—	—	—	0000 0000 0000 0000
IEC4	—	—	—	FLT2BIE	—	FLT2AIE	PWM2IE	—	—	—	—	—	—	—	—	FLT1BIE	0000 0000 0000 0000
IPC18	—	—	—	—	—	FLT2AIP<2:0>			—	PWM2IP<2:0>			—	—	—	—	0000 0100 0100 0000
IPC19	—	—	—	—	—	—	—	—	—	—	—	—	—	FLT2BIP<2:0>			0000 0000 0000 0100
P2TCON	PTEN	—	PTSIDL	—	—	—	—	—	PTOPS<3:0>				PTCKPS<1:0>		PTMOD<1:0>		0000 0000 0000 0000
P2TMR	PTDIR	PWM Timer Count Value Register															0000 0000 0000 0000
P2TPER	—	PWM Time Base Period Register															0000 0000 0000 0000
P2SECMP	SEVTDIR	PWM Special Event Compare Register															0000 0000 0000 0000
PWM2CON1	—	—	—	—	—	—	—	PMOD1	—	—	—	PEN1H	—	—	—	PEN1L	0000 0000 000y 000y[1]
PWM2CON2	—	—	—	—	SEVOPS<3:0>				—	—	—	—	—	IUE	OSYNC	UDIS	0000 0000 0000 0000
P2DTCON1	DTBPS<1:0>		DTB<5:0>						DTAPS<1:0>		DTA<5:0>						0000 0000 0000 0000
P2DTCON2	—	—	—	—	—	—	—	—	—	—	—	—	—	—	DTS1A	DTS1I	0000 0000 0000 0000
P2FLTACON	—	—	—	—	—	—	FAOV1H	FAOV1L	FLTAM	—	—	—	—	—	—	FAEN1	0000 0000 0000 000y[2]
P2OVDCON	—	—	—	—	—	—	POVD1H	POVD1L	—	—	—	—	—	—	POUT1H	POUT1L	1111 1111 0000 0000
P2DC1	PWM Duty Cycle 1 Register																0000 0000 0000 0000

PWM setting

Equation 14-1: PWM Period Calculation for Free Running Count Mode (PTMOD = 00 or 01)

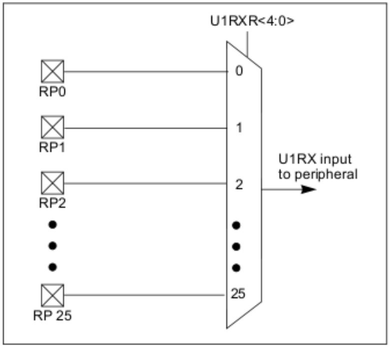
$$PxTPER = \frac{F_{CY}}{F_{PWM} \times (PxTMR\ Prescaler)} - 1$$

Remapable pins

- dynamic assignment of I/O ports (peripherals) to physical ports `RPx`
- for input: input function has register, pin address is assigned
- for output: physical pin has register, function code is assigned
- secured: (un)locking sequence, fuse

Remapable input pins

FIGURE 11-2: REMAPPABLE MUX INPUT FOR U1RX



- RPx is used as an input pin for peripheral pin `Func`
- set in `RPINRy`, bits `FuncR`

Input remapping

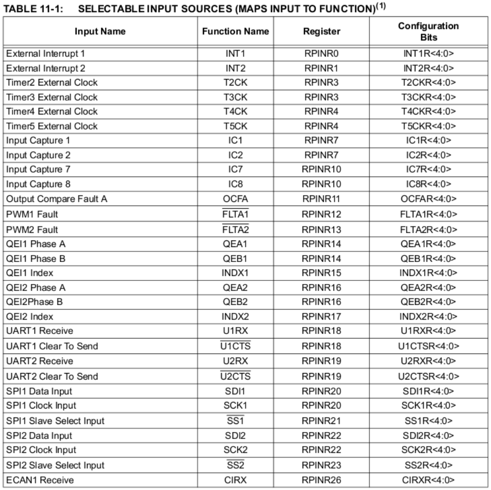


TABLE 11-1: SELECTABLE INPUT SOURCES (MAPS INPUT TO FUNCTION)[1]

Input Name	Function Name	Register	Configuration Bits
External Interrupt 1	INT1	RPINR0	INT1R<4:0>
External Interrupt 2	INT2	RPINR1	INT2R<4:0>
Timer2 External Clock	T2CK	RPINR3	T2CKR<4:0>
Timer3 External Clock	T3CK	RPINR3	T3CKR<4:0>
Timer4 External Clock	T4CK	RPINR4	T4CKR<4:0>
Timer5 External Clock	T5CK	RPINR4	T5CKR<4:0>
Input Capture 1	IC1	RPINR7	IC1R<4:0>
Input Capture 2	IC2	RPINR7	IC2R<4:0>
Input Capture 7	IC7	RPINR10	IC7R<4:0>
Input Capture 8	IC8	RPINR10	IC8R<4:0>
Output Compare Fault A	OCFA	RPINR11	OCFAR<4:0>
PWM1 Fault	$\overline{\text{FLTA1}}$	RPINR12	FLTA1R<4:0>
PWM2 Fault	$\overline{\text{FLTA2}}$	RPINR13	FLTA2R<4:0>
QEI1 Phase A	QEA1	RPINR14	QEA1R<4:0>
QEI1 Phase B	QEB1	RPINR14	QEB1R<4:0>
QEI1 Index	INDX1	RPINR15	INDX1R<4:0>
QEI2 Phase A	QEA2	RPINR16	QEA2R<4:0>
QEI2Phase B	QEB2	RPINR16	QEB2R<4:0>
QEI2 Index	INDX2	RPINR17	INDX2R<4:0>
UART1 Receive	U1RX	RPINR18	U1RXR<4:0>
UART1 Clear To Send	$\overline{\text{U1CTS}}$	RPINR18	U1CTSR<4:0>
UART2 Receive	U2RX	RPINR19	U2RXR<4:0>
UART2 Clear To Send	$\overline{\text{U2CTS}}$	RPINR19	U2CTSR<4:0>
SPI1 Data Input	SDI1	RPINR20	SDI1R<4:0>
SPI1 Clock Input	SCK1	RPINR20	SCK1R<4:0>
SPI1 Slave Select Input	$\overline{\text{SS1}}$	RPINR21	SS1R<4:0>
SPI2 Data Input	SDI2	RPINR22	SDI2R<4:0>
SPI2 Clock Input	SCK2	RPINR22	SCK2R<4:0>
SPI2 Slave Select Input	$\overline{\text{SS2}}$	RPINR23	SS2R<4:0>
ECAN1 Receive	CIRX	RPINR26	CIRXR<4:0>

Input remapping registers

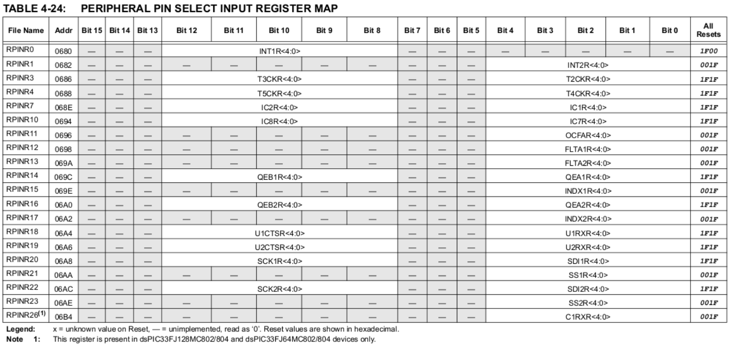


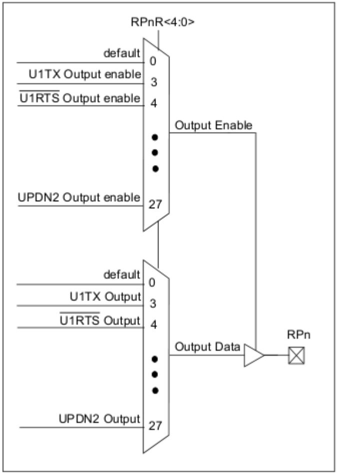
TABLE 4-24: PERIPHERAL PIN SELECT INPUT REGISTER MAP

File Name	Addr	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	All Resets
RPINR0	0680	—	—	—			INT1R<4:0>			—	—	—	—	—	—	—	—	*1F00*
RPINR1	0682	—	—	—	—	—	—	—	—	—	—	—			INT2R<4:0>			*001F*
RPINR3	0686	—	—	—			T3CKR<4:0>			—	—	—			T2CKR<4:0>			*1F1F*
RPINR4	0688	—	—	—			T5CKR<4:0>			—	—	—			T4CKR<4:0>			*1F1F*
RPINR7	068E	—	—	—			IC2R<4:0>			—	—	—			IC1R<4:0>			*1F1F*
RPINR10	0694	—	—	—			IC8R<4:0>			—	—	—			IC7R<4:0>			*1F1F*
RPINR11	0696	—	—	—	—	—	—	—	—	—	—	—			OCFAR<4:0>			*001F*
RPINR12	0698	—	—	—	—	—	—	—	—	—	—	—			FLTA1R<4:0>			*001F*
RPINR13	069A	—	—	—	—	—	—	—	—	—	—	—			FLTA2R<4:0>			*001F*
RPINR14	069C	—	—	—			QEB1R<4:0>			—	—	—			QEA1R<4:0>			*1F1F*
RPINR15	069E	—	—	—	—	—	—	—	—	—	—	—			INDX1R<4:0>			*001F*
RPINR16	06A0	—	—	—			QEB2R<4:0>			—	—	—			QEA2R<4:0>			*1F1F*
RPINR17	06A2	—	—	—	—	—	—	—	—	—	—	—			INDX2R<4:0>			*001F*
RPINR18	06A4	—	—	—			U1CTSR<4:0>			—	—	—			U1RXR<4:0>			*1F1F*
RPINR19	06A6	—	—	—			U2CTSR<4:0>			—	—	—			U2RXR<4:0>			*1F1F*
RPINR20	06A8	—	—	—			SCK1R<4:0>			—	—	—			SDI1R<4:0>			*1F1F*
RPINR21	06AA	—	—	—	—	—	—	—	—	—	—	—			SS1R<4:0>			*001F*
RPINR22	06AC	—	—	—			SCK2R<4:0>			—	—	—			SDI2R<4:0>			*1F1F*
RPINR23	06AE	—	—	—	—	—	—	—	—	—	—	—			SS2R<4:0>			*001F*
RPINR26[1]	06B4	—	—	—	—	—	—	—	—	—	—	—			C1RXR<4:0>			*001F*

Legend: x = unknown value on Reset, — = unimplemented, read as '0'. Reset values are shown in hexadecimal.
Note 1: This register is present in dsPIC33FJ128MC802/804 and dsPIC33FJ64MC802/804 devices only.

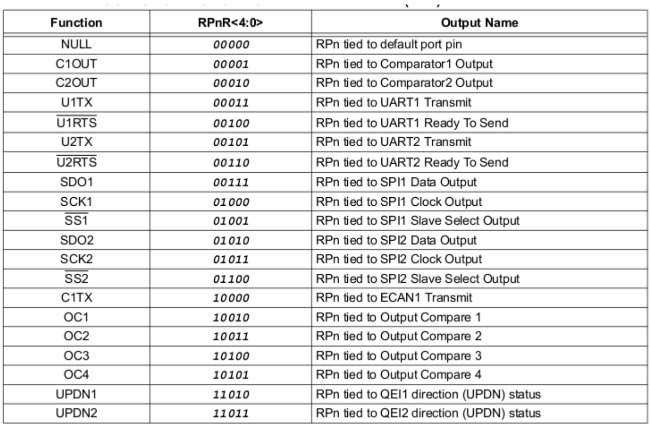
Remapable output pins

FIGURE 11-3: MULTIPLEXING OF REMAPPABLE OUTPUT FOR RPn



- peripheral pin `Func` with code is used as an output pin `RPx`
- set in `RPOUTRy`, bits `RPxR`

Output remapping



Function	RPnR<4:0>	Output Name
NULL	*00000*	RPn tied to default port pin
C1OUT	*00001*	RPn tied to Comparator1 Output
C2OUT	*00010*	RPn tied to Comparator2 Output
U1TX	*00011*	RPn tied to UART1 Transmit
$\overline{\text{U1RTS}}$	*00100*	RPn tied to UART1 Ready To Send
U2TX	*00101*	RPn tied to UART2 Transmit
$\overline{\text{U2RTS}}$	*00110*	RPn tied to UART2 Ready To Send
SDO1	*00111*	RPn tied to SPI1 Data Output
SCK1	*01000*	RPn tied to SPI1 Clock Output
$\overline{\text{SS1}}$	*01001*	RPn tied to SPI1 Slave Select Output
SDO2	*01010*	RPn tied to SPI2 Data Output
SCK2	*01011*	RPn tied to SPI2 Clock Output
$\overline{\text{SS2}}$	*01100*	RPn tied to SPI2 Slave Select Output
C1TX	*10000*	RPn tied to ECAN1 Transmit
OC1	*10010*	RPn tied to Output Compare 1
OC2	*10011*	RPn tied to Output Compare 2
OC3	*10100*	RPn tied to Output Compare 3
OC4	*10101*	RPn tied to Output Compare 4
UPDN1	*11010*	RPn tied to QEI1 direction (UPDN) status
UPDN2	*11011*	RPn tied to QEI2 direction (UPDN) status

Output remapping registers

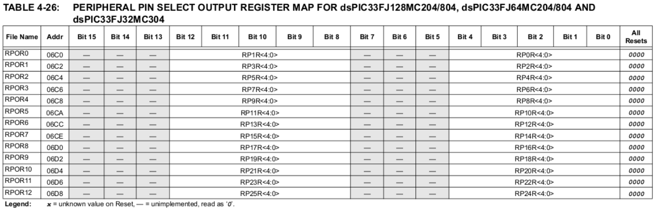


TABLE 4-26: PERIPHERAL PIN SELECT OUTPUT REGISTER MAP FOR dsPIC33FJ128MC204/804, dsPIC33FJ64MC204/804 AND dsPIC33FJ32MC304

File Name	Addr	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	All Resets
RPOR0	06C0	—	—	—	RP1R<4:0>					—	—	—	RP0R<4:0>					*0000*
RPOR1	06C2	—	—	—	RP3R<4:0>					—	—	—	RP2R<4:0>					*0000*
RPOR2	06C4	—	—	—	RP5R<4:0>					—	—	—	RP4R<4:0>					*0000*
RPOR3	06C6	—	—	—	RP7R<4:0>					—	—	—	RP6R<4:0>					*0000*
RPOR4	06C8	—	—	—	RP9R<4:0>					—	—	—	RP8R<4:0>					*0000*
RPOR5	06CA	—	—	—	RP11R<4:0>					—	—	—	RP10R<4:0>					*0000*
RPOR6	06CC	—	—	—	RP13R<4:0>					—	—	—	RP12R<4:0>					*0000*
RPOR7	06CE	—	—	—	RP15R<4:0>					—	—	—	RP14R<4:0>					*0000*
RPOR8	06D0	—	—	—	RP17R<4:0>					—	—	—	RP16R<4:0>					*0000*
RPOR9	06D2	—	—	—	RP19R<4:0>					—	—	—	RP18R<4:0>					*0000*
RPOR10	06D4	—	—	—	RP21R<4:0>					—	—	—	RP20R<4:0>					*0000*
RPOR11	06D6	—	—	—	RP23R<4:0>					—	—	—	RP22R<4:0>					*0000*
RPOR12	06D8	—	—	—	RP25R<4:0>					—	—	—	RP24R<4:0>					*0000*

Legend: *x* = unknown value on Reset, — = unimplemented, read as '*0*'.

Remapping security

```
//unlock procedure
OSCCON = 0x46;
OSCCON = 0x57;
OSCCONbits.IOLOCK = 0;

// some manipulation
RPOR10bits.RP21R = 0x3;
RPINR18bits.U1RXR = 20;

// lock procedure
OSCCON = 0x46;
OSCCON = 0x57;
OSCCONbits.IOLOCK = 1;
```

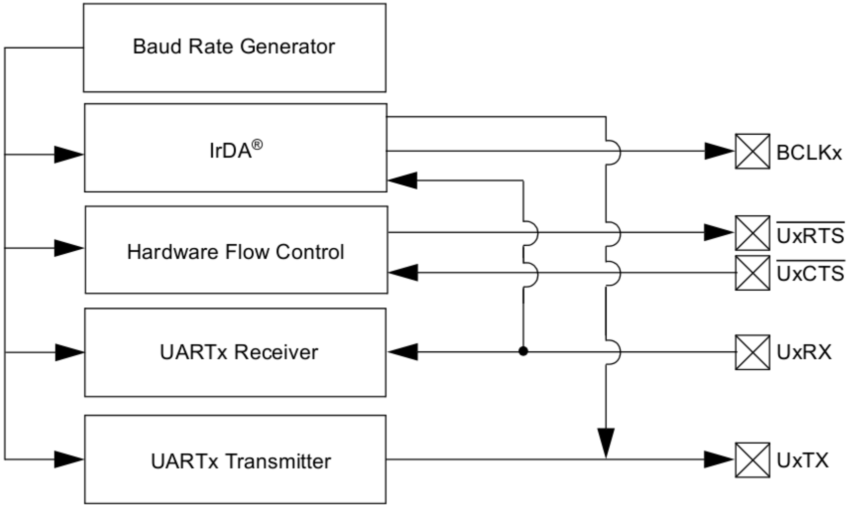
Remapping security II.

```
// allow multiple port remapping
_FOSC(  ... & IOL1WAY_OFF )

// allow multiple port just once between two resets
_FOSC(  ... & IOL1WAY_ON )
```

Ex. Remapping security II.

```
// remapping input
// assign RP20 to UART1 RX
RPINR18bits.U1RXR = 20;

// remapping output
// assign function of UART1 TX to port RP21
RPOR10bits.RP21R = 0x3;
```

Universal Asynchroneous Receiver-Transmitter



Baud-rate Generator

Equation 17-1: UART Baud Rate (BRGH = 0)

$$Baud\ Rate = \frac{F_{CY}}{16 \times (UxBRG + 1)} \quad \text{......(1)}$$

$$UxBRG = \frac{F_{CY}}{16 \times Baud\ Rate} - 1 \quad \text{......(2)}$$

Note: F_{CY} denotes the instruction cycle clock frequency (FOSC/2).

Equation 17-2: UART Baud Rate (BRGH = 1)

$$Baud\ Rate = \frac{F_{CY}}{4 \times (UxBRG + 1)} \quad \text{......(1)}$$

$$UxBRG = \frac{F_{CY}}{4 \times Baud\ Rate} - 1 \quad \text{......(2)}$$

Note: F_{CY} denotes the instruction cycle clock frequency.

UART Registers

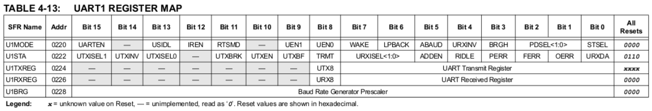


TABLE 4-13: UART1 REGISTER MAP

SFR Name	Addr	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	All Resets
U1MODE	0220	UARTEN	—	USIDL	IREN	RTSMD	—	UEN1	UEN0	WAKE	LPBACK	ABAUD	URXINV	BRGH	PDSEL<1:0>		STSEL	*0000*
U1STA	0222	UTXISEL1	UTXINV	UTXISEL0	—	UTXBRK	UTXEN	UTXBF	TRMT	URXISEL<1:0>		ADDEN	RIDLE	PERR	FERR	OERR	URXDA	*0110*
U1TXREG	0224	—	—	—	—	—	—	—	UTX8	UART Transmit Register								***xxxx***
U1RXREG	0226	—	—	—	—	—	—	—	URX8	UART Received Register								*0000*
U1BRG	0228	Baud Rate Generator Prescaler																*0000*

Legend: ***x*** = unknown value on Reset, — = unimplemented, read as '*0*'. Reset values are shown in hexadecimal.

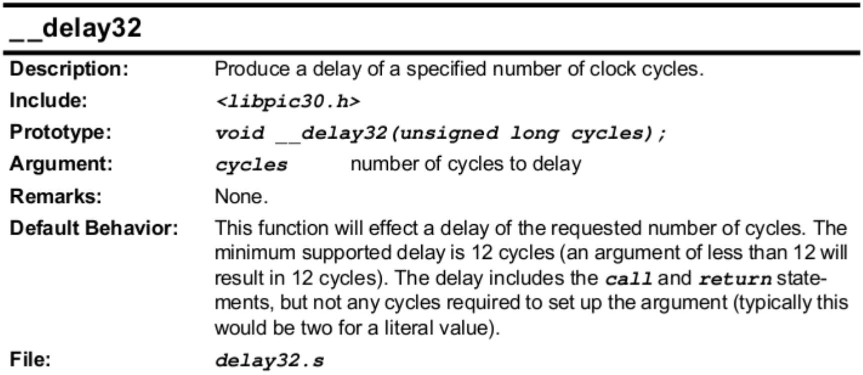
UART initialization

```
TRISCbits.TRISC4 = 1; // uart1 RX input (RP20)
TRISCbits.TRISC5 = 0; // uart1 TX output (RP21)

// reg UIMODE, see dsh p216
U1MODEbits.UARTEN = 0; // stop it now

U1MODEbits.STSEL = 0; // 1 stop bit
U1MODEbits.PDSEL = 0b00; // 8 bit, no parity
U1MODEbits.BRGH = 0; // 16 clocks per bit period

U1BRG = 64; // (10e6 / (4 * 9600)) - 1
U1STA = 0; // clear status reg

OSCCON = 0x46; OSCCON = 0x57;
OSCCONbits.IOLOCK = 0;

RPOR10bits.RP21R = 0x3;
RPINR18bits.U1RXR = 20;

OSCCON = 0x46; OSCCON = 0x57;
OSCCONbits.IOLOCK = 1;

U1MODEbits.UARTEN = 1; // enable UART peripheral
U1STAbits.UTXEN = 1; // enable TX
```

UART use

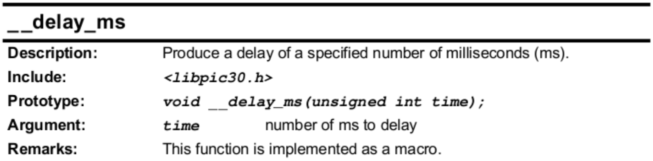
```
void UART_PutChar(char c){
  while (U1STAbits.UTXBF); // Wait for space in UART2 Tx buffer
  U1TXREG = c; // Write character to UART2
}

int UART_GetChar(){
  // buffer overrun error
  if(U1STAbits.OERR == 1){
    U1STAbits.OERR = 0;
    return -1;
  }

  while(U1STAbits.URXDA == 0);
  a = U1RXREG;
  return a;
}
```

Delay Functions



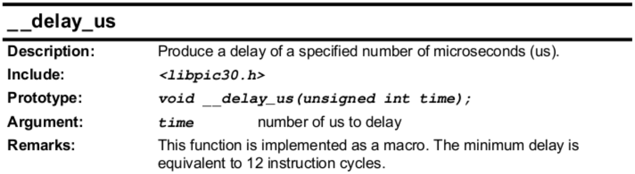
__delay32

Description:	Produce a delay of a specified number of clock cycles.
Include:	`<libpic30.h>`
Prototype:	`void __delay32(unsigned long cycles);`
Argument:	`cycles` number of cycles to delay
Remarks:	None.
Default Behavior:	This function will effect a delay of the requested number of cycles. The minimum supported delay is 12 cycles (an argument of less than 12 will result in 12 cycles). The delay includes the `call` and `return` statements, but not any cycles required to set up the argument (typically this would be two for a literal value).
File:	`delay32.s`



__delay_ms

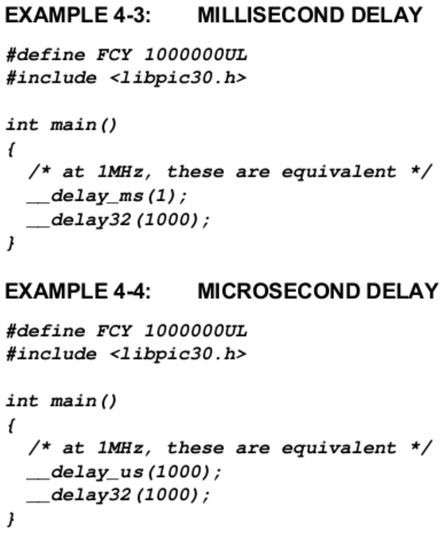
Description:	Produce a delay of a specified number of milliseconds (ms).
Include:	`<libpic30.h>`
Prototype:	`void __delay_ms(unsigned int time);`
Argument:	*time* number of ms to delay
Remarks:	This function is implemented as a macro.



__delay_us

Description:	Produce a delay of a specified number of microseconds (us).
Include:	`<libpic30.h>`
Prototype:	`void __delay_us(unsigned int time);`
Argument:	*time* number of us to delay
Remarks:	This function is implemented as a macro. The minimum delay is equivalent to 12 instruction cycles.

Delay Functions - examples



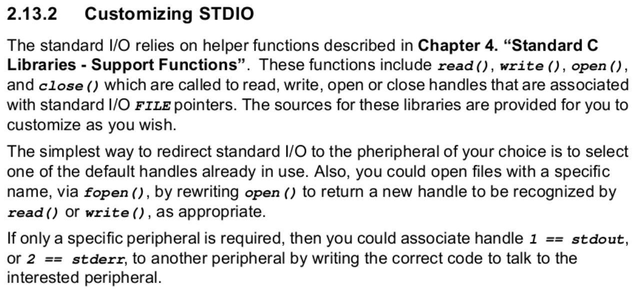
EXAMPLE 4-3: MILLISECOND DELAY

```
#define FCY 1000000UL
#include <libpic30.h>

int main()
{
  /* at 1MHz, these are equivalent */
  __delay_ms(1);
  __delay32(1000);
}
```

EXAMPLE 4-4: MICROSECOND DELAY

```
#define FCY 1000000UL
#include <libpic30.h>

int main()
{
  /* at 1MHz, these are equivalent */
  __delay_us(1000);
  __delay32(1000);
}
```

Redirecting stdio



2.13.2 Customizing STDIO

The standard I/O relies on helper functions described in **Chapter 4. "Standard C Libraries - Support Functions"**. These functions include ***read()***, ***write()***, ***open()***, and ***close()*** which are called to read, write, open or close handles that are associated with standard I/O ***FILE*** pointers. The sources for these libraries are provided for you to customize as you wish.

The simplest way to redirect standard I/O to the pheripheral of your choice is to select one of the default handles already in use. Also, you could open files with a specific name, via ***fopen()***, by rewriting ***open()*** to return a new handle to be recognized by ***read()*** or ***write()***, as appropriate.

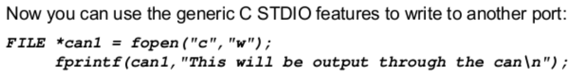
If only a specific peripheral is required, then you could associate handle ***1 == stdout***, or ***2 == stderr***, to another peripheral by writing the correct code to talk to the interested peripheral.

```
/* should be in a header file */
enum my_handles {
    handle_stdin,
    handle_stdout,
    handle_stderr,
    handle_can1,
    handle_can2,
    handle_spi1,
    handle_spi2,
};

int __attribute__((__weak__, __section__(".libc"))) open(const char
*name, int access, int mode) {
     switch (name[0]) {
       case 'i' : return handle_stdin;
       case 'o' : return handle_stdout;
       case 'e' : return handle_stderr;
       case 'c' : return handle_can1;
       case 'C' : return handle_can2;
       case 's' : return handle_spi1;
       case 'S' : return handle_spi2;
       default: return handle_stderr;
    }
}
```

In *write()*, you would write:

```
write(int handle, void *buffer, unsigned int len) {
     int i;
     volatile UxMODEBITS *umode = &U1MODEbits;
     volatile UxSTABITS *ustatus = &U1STAbits;
     volatile unsigned int *txreg = &U1TXREG;
     volatile unsigned int *brg = &U1BRG;

     switch (handle)


       {
       default:
       case 0:
       case 1:
       case 2:
          if ((__C30_UART != 1) && (&U2BRG)) {
             umode = &U2MODEbits;
             ustatus = &U2STAbits;
             txreg = &U2TXREG;
             brg = &U2BRG;
          }
          if ((umode->UARTEN) == 0)
          {
             *brg = 0;
             umode->UARTEN = 1;
          }
          if ((ustatus->UTXEN) == 0)
          {
             ustatus->UTXEN = 1;
          }
          for (i = len; i; --i)
          {
             while ((ustatus->TRMT) ==0);
             *txreg = *(char*)buffer++;
          }
          break;
       case handle_can1:  /* code to support can1 */
          break;
       case handle_can2:  /* code to support can2 */
          break;
       case handle_spi1:  /* code to support spi1 */
          break;
       case handle_spi2:  /* code to support spi2 */
          break;
       }
       return(len);
  }
```

Ex. STDIO redirection



Now you can use the generic C STDIO features to write to another port:

```
FILE *can1 = fopen("c","w");
     fprintf(can1,"This will be output through the can\n");
```

Ex. STDIO redirection

```
#include <libpic30.h>      /* a new header file for these defintions */
#include <stdio.h>
#include <p33Fxxxx.h>

_FOSCSEL ( FNOSC_PRI );     //External XTAL
_FOSC( POSCMD_HS & IOL1WAY_OFF );         // HS mode, 10-40MHZ + IO mapping lock
_FWDT(FWDTEN_OFF);

main() {

 __C30_UART=1;

  OSCCON = 0x46;
  OSCCON = 0x57;
  OSCCONbits.IOLOCK = 0;

  RPOR10bits.RP21R = 0x3;
  RPINR18bits.U1RXR = 20;

  OSCCON = 0x46;
  OSCCON = 0x57;
  OSCCONbits.IOLOCK = 1;
```

Ex. STDIO redirection II.

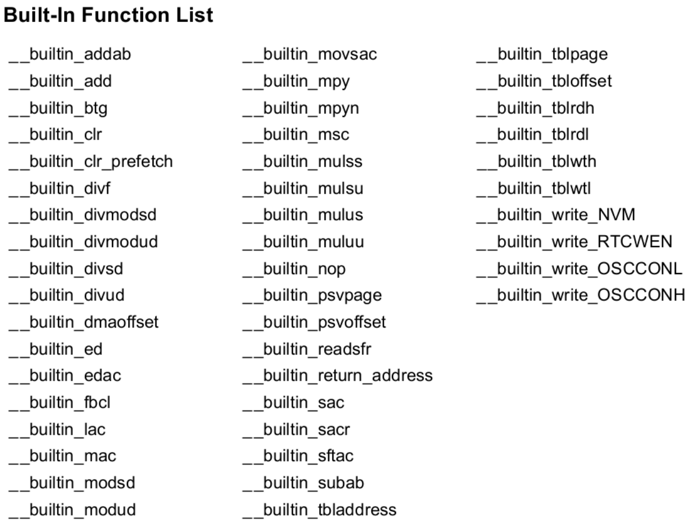
```
   TRISCbits.TRISC4 = 1;   // uart1 RX input (RP20)
   TRISCbits.TRISC5 = 0;   // uart1 TX output (RP21)

   U1BRG = 64;
   U1MODEbits.BRGH = 1;
   U1MODEbits.UARTEN = 1;

  while (1) {
    __builtin_btg(&LATA,6);
    printf("Hello world %d\n",U1BRG);
  }

 }
```

C30 Built-in functions

Built-in functions give the C programmer access to assembler operators or machine instructions that are currently only accessible using inline assembly, but are sufficiently useful that they are applicable to a broad range of applications. Built-in functions are coded in C source files syntactically like function calls, but they are compiled to assembly code that directly implements the function, and do not involve function calls or library routines.
Why built-in functions is preferable to inline assembly:

- 1. Providing built-in functions for specific purposes simplifies coding.
- 2. Certain optimizations are disabled when inline assembly is used. This is not the case for built-in functions.
- 3. For machine instructions that use dedicated registers, coding inline assembly while avoiding register allocation errors can require considerable care. The built-in functions make this process simpler as you do not need to be concerned with the particular register requirements for each individual machine instruction.

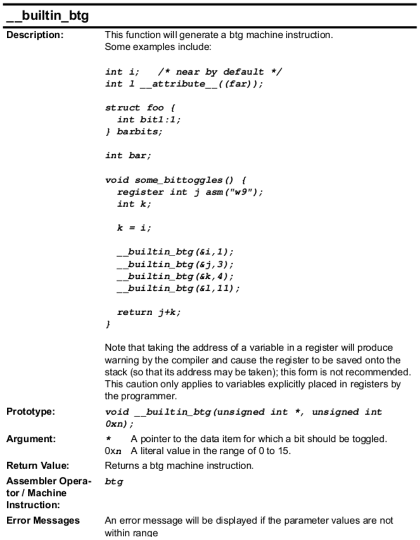
List of Built-in Functions



Built-In Function List

__builtin_addab	__builtin_movsac	__builtin_tblpage
__builtin_add	__builtin_mpy	__builtin_tbloffset
__builtin_btg	__builtin_mpyn	__builtin_tblrdh
__builtin_clr	__builtin_msc	__builtin_tblrdl
__builtin_clr_prefetch	__builtin_mulss	__builtin_tblwth
__builtin_divf	__builtin_mulsu	__builtin_tblwtl
__builtin_divmodsd	__builtin_mulus	__builtin_write_NVM
__builtin_divmodud	__builtin_muluu	__builtin_write_RTCWEN
__builtin_divsd	__builtin_nop	__builtin_write_OSCCONL
__builtin_divud	__builtin_psvpage	__builtin_write_OSCCONH
__builtin_dmaoffset	__builtin_psvoffset	
__builtin_ed	__builtin_readsfr	
__builtin_edac	__builtin_return_address	
__builtin_fbcl	__builtin_sac	
__builtin_lac	__builtin_sacr	
__builtin_mac	__builtin_sftac	
__builtin_modsd	__builtin_subab	
__builtin_modud	__builtin_tbladdress	

Ex. Built-in Functions



__builtin_btg

Description: This function will generate a btg machine instruction. Some examples include:

```
int i;    /* near by default */
int l __attribute__((far));

struct foo {
  int bit1:1;
} barbits;

int bar;

void some_bittoggles() {
  register int j asm("w9");
  int k;

  k = i;

  __builtin_btg(&i,1);
  __builtin_btg(&j,3);
  __builtin_btg(&k,4);
  __builtin_btg(&l,11);

  return j+k;
}
```

Note that taking the address of a variable in a register will produce warning by the compiler and cause the register to be saved onto the stack (so that its address may be taken); this form is not recommended. This caution only applies to variables explicitly placed in registers by the programmer.

Prototype: `void __builtin_btg(unsigned int *, unsigned int 0xn);`

Argument:

`*`	A pointer to the data item for which a bit should be toggled.
0x*n*	A literal value in the range of 0 to 15.

Return Value: Returns a btg machine instruction.

Assembler Operator / Machine Instruction: *btg*

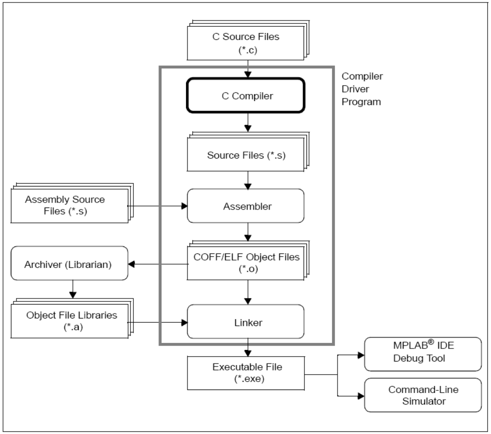
Error Messages An error message will be displayed if the parameter values are not within range

Power Saving

- `Sleep()` – all shut down, interrupt possible
- `Idle()` – peripherals operating

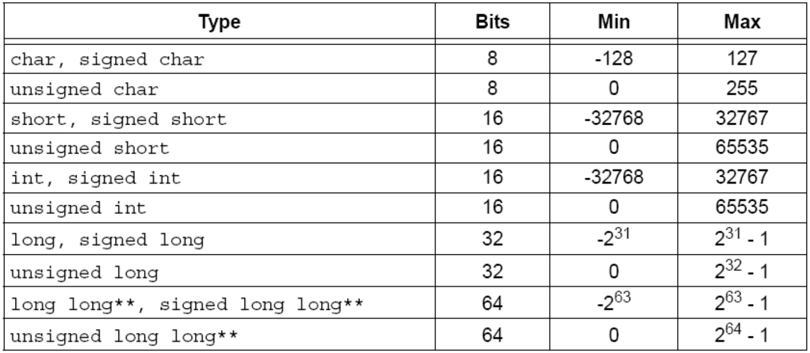
Compiler workflow

FIGURE 1-1: SOFTWARE DEVELOPMENT TOOLS DATA FLOW



Integer Data Types

TABLE 5-1: INTEGER DATA TYPES



Type	Bits	Min	Max
`char, signed char`	8	-128	127
`unsigned char`	8	0	255
`short, signed short`	16	-32768	32767
`unsigned short`	16	0	65535
`int, signed int`	16	-32768	32767
`unsigned int`	16	0	65535
`long, signed long`	32	-2^{31}	$2^{31} - 1$
`unsigned long`	32	0	$2^{32} - 1$
`long long**, signed long long**`	64	-2^{63}	$2^{63} - 1$
`unsigned long long**`	64	0	$2^{64} - 1$

** ANSI-89 extension

Real Data Types

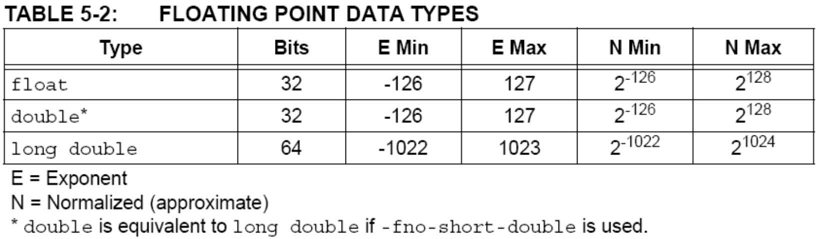


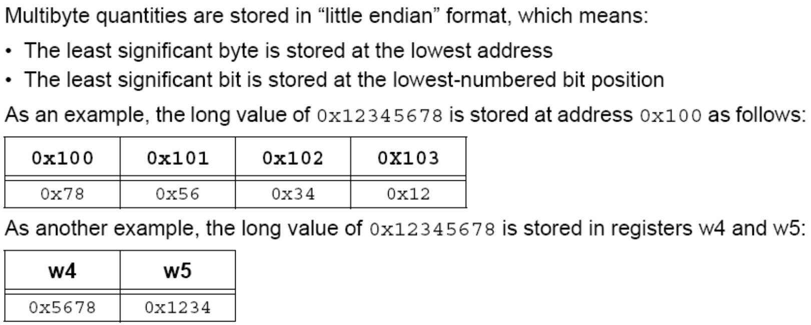
TABLE 5-2: FLOATING POINT DATA TYPES

Type	Bits	E Min	E Max	N Min	N Max
`float`	32	-126	127	2^{-126}	2^{128}
`double`*	32	-126	127	2^{-126}	2^{128}
`long double`	64	-1022	1023	2^{-1022}	2^{1024}

E = Exponent
N = Normalized (approximate)
* `double` is equivalent to `long double` if `-fno-short-double` is used.

Endianity

dsPIC33 family is little-endian



Multibyte quantities are stored in "little endian" format, which means:

- The least significant byte is stored at the lowest address
- The least significant bit is stored at the lowest-numbered bit position

As an example, the long value of `0x12345678` is stored at address `0x100` as follows:

0x100	0x101	0x102	0X103
0x78	0x56	0x34	0x12

As another example, the long value of `0x12345678` is stored in registers w4 and w5:

w4	w5
0x5678	0x1234

Misc

- device support files
- built-in functions
- standard library
- assembler mixing