TSK165x RISC MCU

Summary

Core Reference
CR0114 (v2.0) March 13, 2008

The TSK165x is a fully functional, 8-bit controller that employs RISC architecture with a streamlined set of single word instructions. This core reference includes architectural and hardware descriptions, instruction sets and on-chip debugging functionality for the TSK165x family.

The TSK165x is instruction set compatible with the PIC16C5X family. All instructions are single cycle, except for program branches which take two cycles.

Important Notice: Supply of this soft core under the terms and conditions of the Altium End-User License Agreement does not convey nor imply any patent rights to the supplied technologies. Users are cautioned that a license may be required for any use covered by such patent rights.

Features

- RISC Control Unit
  - Instruction set – comprising all single-word instructions (31 in total)
  - 12-bit instruction decoder
  - Single cycle instruction execution (except double-cycle branch instructions)
  - 7 dedicated special function registers (SFRs)
  - 8-level deep hardware stack
  - Direct, indirect and relative addressing modes for data and instructions
- Arithmetic Logic Unit
  - 8 bit arithmetic operations
  - 8 bit logical operations
  - Boolean manipulations
- Device Reset Timer
- I/O ports
  - TSK165A, TSK165B : 3, 8-bit I/O ports
  - TSK165C : 6, 8-bit I/O ports
- Data Memory interface
  - TSK165A : can address up to 16+9 bytes of Read/Write Data memory space
  - TSK165B, TSK165C : can address up to 64+9 bytes of Read/Write Data memory space
- Program Memory interface
  - TSK165A : can address up to 512 bytes of Program memory Space
  - TSK165B, TSK165C : can address up to 2KB of Program memory Space

Performance

The high performance of the TSK165x can be attributed to a number of architectural features commonly found in RISC microprocessors. To begin with, the TSK165x uses a Harvard architecture in which program and data are accessed on separate buses. This improves bandwidth over the more traditional von Neumann architecture, where program and data are fetched on the same bus. Separating Program and Data memory allows instructions to be sized at 12 bits, rather than the 8-bits used for data.

12-bits wide instruction opcodes make it possible to have all single word instructions. A 12-bit wide Program memory access bus fetches a 12-bit instruction in a single cycle. A two-stage pipeline overlaps the fetch and execution of instructions. Consequently, all instructions (31 in total) execute in a single cycle except for program branches.

Available Devices

Three variants of the standard microcontroller core are available – the TSK165A, TSK165B and TSK165C respectively. Table 1 summarizes the key differences between the three.

Table 1. TSK165x standard core variants

Feature	TSK165A	TSK165B	TSK165C
Addressable Program memory	512*12	2K*12	Same features as the TSK165B, plus three additional 8-bit ports.
Data memory	16 + 9 bytes	64 + 9 bytes	
Program Counter width	9 bits	11 bits	
Stack width	9 bits	11 bits	

In addition, a corresponding debug-enabled (OCD) version of each variant is also available (TSK165A_D, TSK165B_D and TSK165C_D respectively).

Note: Throughout this document, differences between core variants are listed in terms of the standard core devices (TSK165A/B/C). Unless specified otherwise, the feature/description applies to the debug-enabled version of the variant (TSK165A_D/B_D/C_D) in exactly the same way.

All devices in the TSK165x family can be found in the FPGA Processors integrated library (`FPGA Processors.IntLib`), located in the `\Library\Fpga` folder of the installation.

Architectural Overview

Symbols

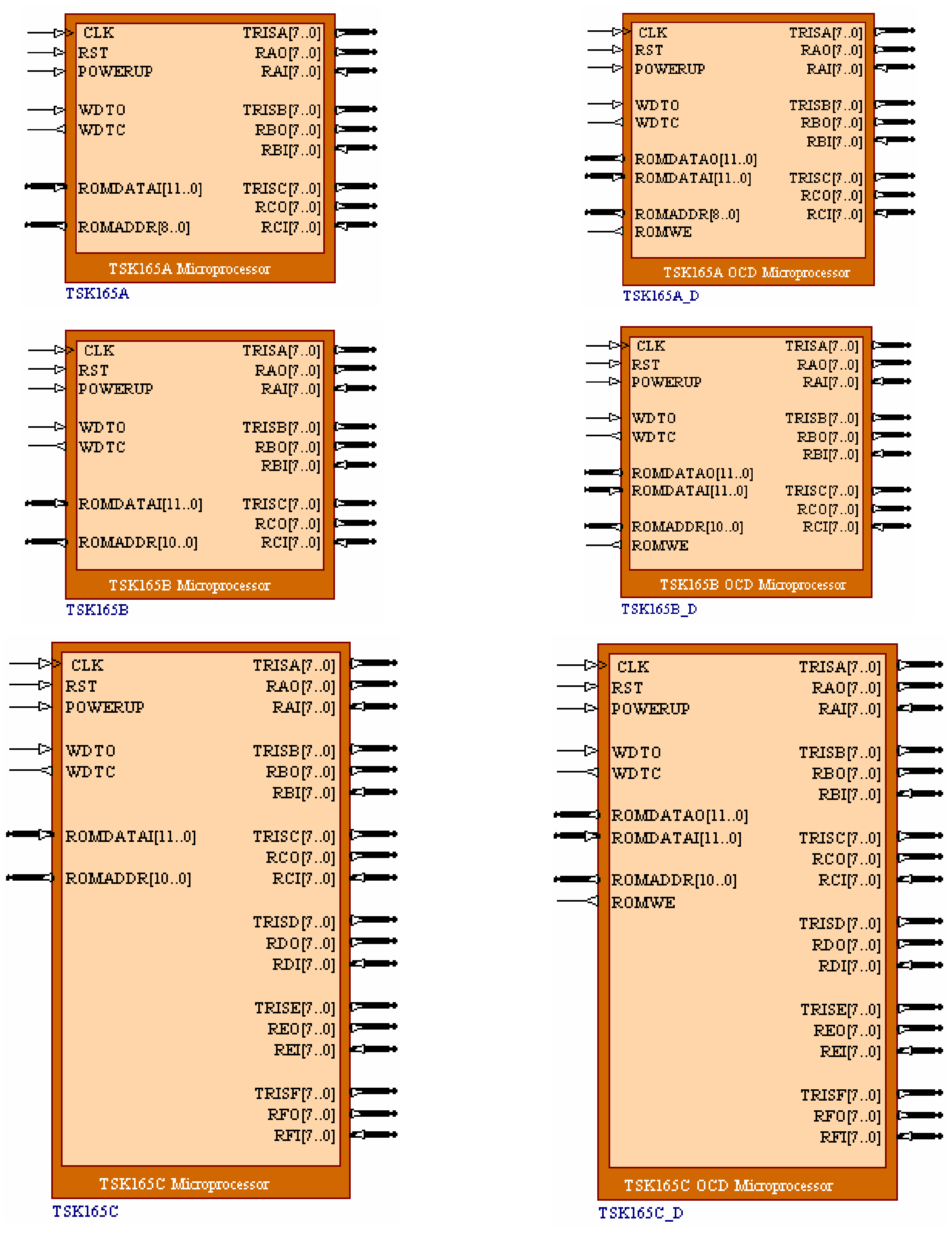


Figure 1. TSK165x family symbols

Pin Description

The pinout of the TSK165x has not been fixed to any specific device I/O - allowing flexibility with user application. The TSK165x contains only unidirectional pins (inputs or outputs).

Table 2. TSK165x Pin description

Name	Type	Polarity/Bus size	Description
			Control Signals
CLK	I	Rise	External clock used for internal clock counters and all other synchronous circuitry
RST	I	High	External system reset. A high on this pin while the external system clock (CLK) is running resets the device into the user reset state
Powerup	I	High	Power up reset. A low on this pin while the external system clock (CLK) is running resets the device into the power up reset state
			External Watchdog Timer Interface Signals
WDTO	I	High	Overflow signal from external Watchdog Timer. When a high is received on this pin, the microcontroller is reset.
WDTC	O	High	A high on this pin, provided by the CLRWDT instruction, sends a clear signal to the external Watchdog Timer.
			Program Memory Interface Signals
ROMDATAO[1]	O	12	Memory data bus output
ROMDATAI	I	12	Memory data bus input
ROMADDR	O	11[2]	Memory address bus
ROMWE[1]	O	High	Memory write enable
			I/O Ports Interface Signals
			Port Register A
RAI	I	8	Port A configured as input
RAO	O	8	Port A configured as output
TRISA	O	8	Output driver control register for Port A
			Port Register B
RBI	I	8	Port B configured as input
RBO	O	8	Port B configured as output
TRISB	O	8	Output driver control register for Port B
			Port Register C
RCI	I	8	Port C configured as input
RCO	O	8	Port C configured as output
TRISC	O	8	Output driver control register for Port C
			Port Register D[3]
RDI	I	8	Port D configured as input
RDO	O	8	Port D configured as output
TRISD	O	8	Output driver control register for Port D
			Port Register E[3]
REI	I	8	Port E configured as input
REO	O	8	Port E configured as output
TRISE	O	8	Output driver control register for Port E

[1] TSK165A_D, TSK165B_D and TSK165C_D only

[2] Value shown is for the TSK165B and TSK165C. For the TSK165A, the address will be 9 bits.

[3] TSK165C only

Name	Type	Polarity/Bus size	Description
			Port Register F[3]
RFI	I	8	Port F configured as input
RFO	O	8	Port F configured as output
TRISF	O	8	Output driver control register for Port F

Memory Organization

The TSK165x microcontroller incorporates the Harvard architecture with separate program (code) and data spaces. The two memory areas are organized as follows:

- Program memory is organized into pages, each 512 bytes in size. For the various members in the TSK165x family, the number of pages are as follows:
  - TSK165A (512 bytes) – 1 page
  - TSK165B, TSK165C (2KB) – 4 pages
- For the TSK165B and TSK165C, the pages are accessed using two page select bits of the STATUS register.
- Data memory utilizes a banking scheme. Banks of Data memory are accessed using the File Selection Register (FSR). Note that banking only applies in the case of the TSK165B and TSK165C, where the Data memory space is larger. In the TSK165A, the Data memory space is the equivalent of a single bank in size.

Program Memory

In the TSK165x, the size of the Program Counter (PC) depends on the particular core variant being used.

TSK165A - 9-bit Program Counter capable of addressing 512∗12 Program memory space (Figure 2).

TSK165B, TSK165C - 11-bit Program Counter capable of addressing 2K∗12 Program memory space (Figure 2).

Program memory is read whenever the CPU performs a fetching instruction. Internally, an instruction is fetched from Program memory during every instruction cycle and latched in the fetch register.

After a reset has been issued, the CPU starts program execution from location 000h.

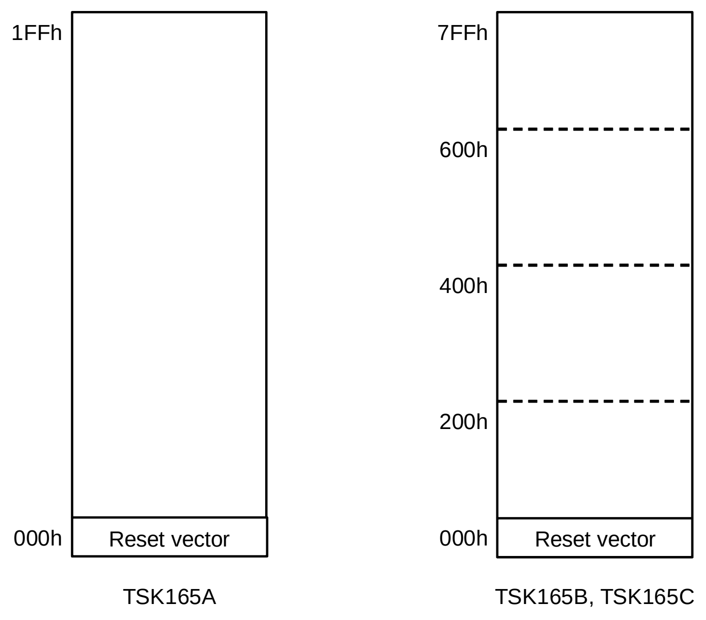


Figure 2. Program memory organization

When using Program memory, a separate block is placed in the design – external to the component symbol for the core. With a standard core variant (TSK165A/B/C), a block of ROM is used, the size of which depends on the requirements of the design. With an OCD variant, because this version of the core allows you to write to Program memory space, RAM must be used instead, as shown in the example of Figure 3.

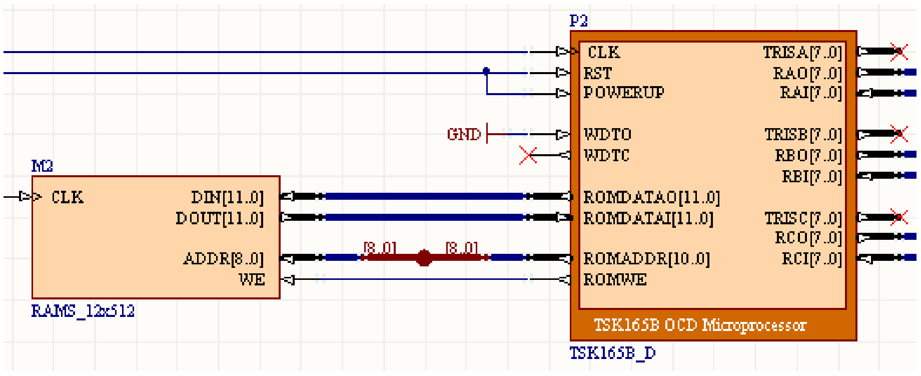


Figure 3. Using RAM for TSK165A_D Program memory

RAM and ROM blocks can be found in the FPGA Memories integrated library (`\Library\Fpga\FPGA Memories.IntLib`).

Program Memory Timing

The execution of instruction N is performed during the fetch of instruction N+1.

Program Memory Read Cycle

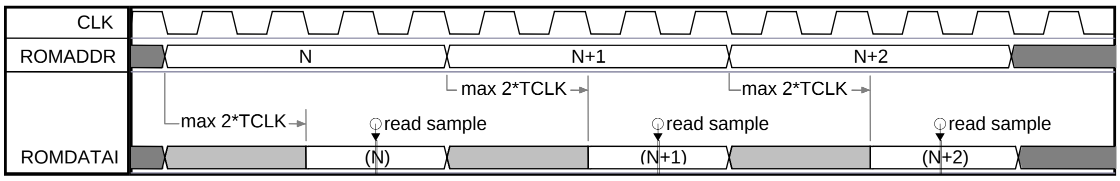


Figure 4. Program memory read cycle

Note:		
	TCLK	- time period of CLK signal
	N	- address of current instruction to be executed
	(N)	- instruction fetched from address N
	N+1	- address of next instruction to be executed
	read sample	- point at which data is read from the bus into the internal fetch register.

Data Memory

Data memory is composed of registers, or bytes of RAM. Therefore, Data memory for a device is specified by its register file. The register file is divided into two functional groups: General Purpose Registers and Special Function Registers.

- The General Purpose Registers (GPRs) are used to store data and control information for use with specified instructions.
- The Special Function Registers (SFRs) are used by the CPU and its peripheral modules to control the operation of the device. These registers include the STATUS Register, the I/O registers and the File Selection Register (FSR). In addition, various special purpose registers are used to control, for example, the I/O port configuration.

The variants of the TSK165x differ in the size of addressable Data memory space they offer:

- In the TSK165A, a 16 byte RAM block is used and the overall Data memory space is comprised of a single bank only (Bank 0). Addressable GPR space is 16 + 9 bytes. The memory space assigned to the SFRs is 7 bytes (see Figure 5)
- In the TSK165B, a 64 byte RAM block is used and the overall Data memory space is comprised of four banks (Bank 0 – Bank 3). Addressable GPR space is 64 + 9 bytes. The memory space assigned to the SFRs is 7 bytes (see Figure 5).

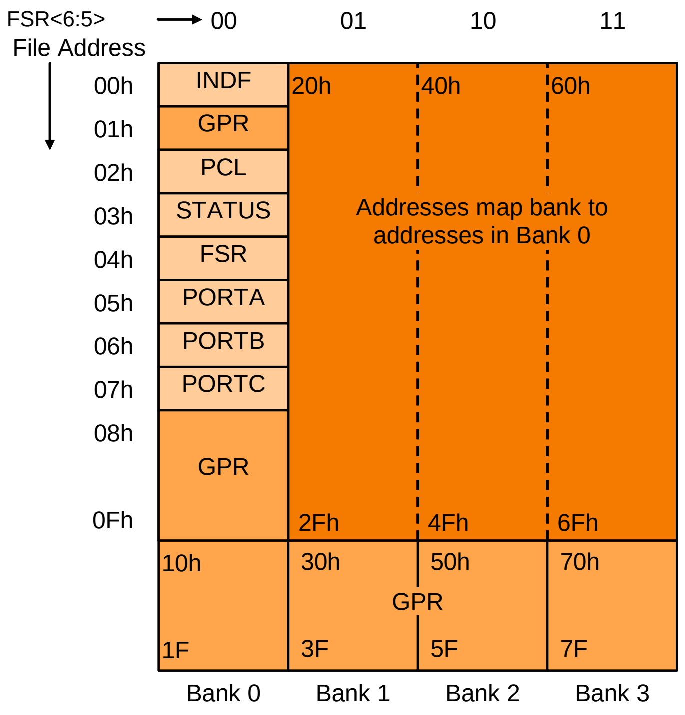


Figure 5. Data memory organization (TSK165A, TSK165B)

- In the TSK165C, a 64 byte RAM block is used and the overall Data memory space is comprised of four banks (Bank 0 – Bank 3). Addressable GPR space is 64 + 6 bytes. The memory space assigned to the SFRs is 10 bytes (see Figure 6).

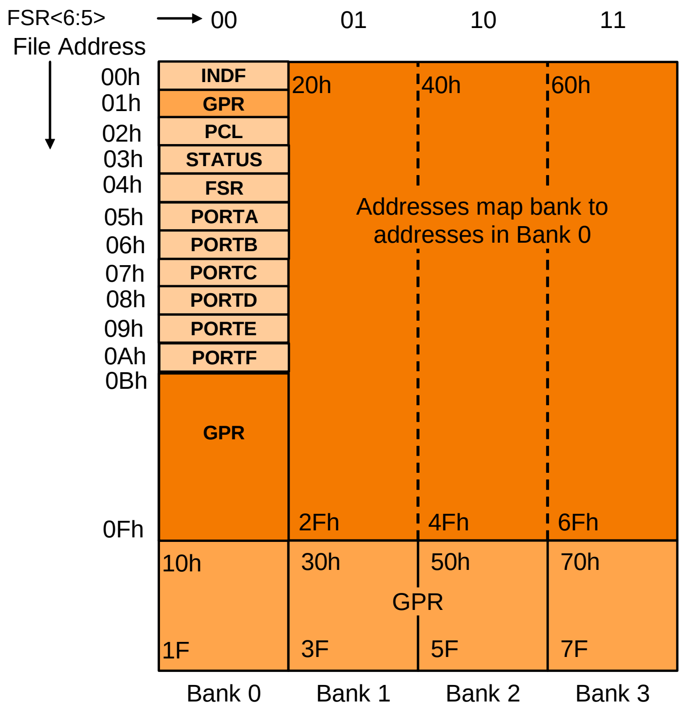


Figure 6. Data memory organization (TSK165C)

As the Data memory space in the TSK165A is not greater than 32 bytes, a banking scheme (selection of banks within the memory organization) is not required and therefore not implemented. In the TSK165B and TSK165C – where the Data memory space is greater – such a banking scheme is implemented through the use of two bank selection bits in the File Select register (FSR). As shown in Figure 5 and Figure 6, these two bits (FSR 6:5) select the banks of memory as follows:

- 00 – Bank 0
- 01 – Bank 1
- 10 – Bank 2

- 11 – Bank 3.

With respect to Banks 1-3, the first 16 bytes map exactly to the first 16 bytes of Bank 0 (the same addresses are seen in these first 16 bytes).

The Data Bus Interface services Data memory when the ramwe signal is active. The TSK165x reads from / writes to Data memory when the CPU executes any of the byte or bit oriented instructions.

As the TSK165x has a dedicated block of Data RAM, the Data memory interface is not exposed to the user through the schematic symbol. As such, the size of the Data memory cannot be upgraded. If, for example, you have used the TSK165A in your design and find that you need more than the 16 +9 bytes of GPR-assigned Data memory space, you would need to either use one of the other variants in the TSK165x family (e.g. the TSK165B, with 64 + 9 bytes) or use a different microcontroller altogether that offered increased Data memory space.

Data Memory Timing

Data Memory Read Cycle

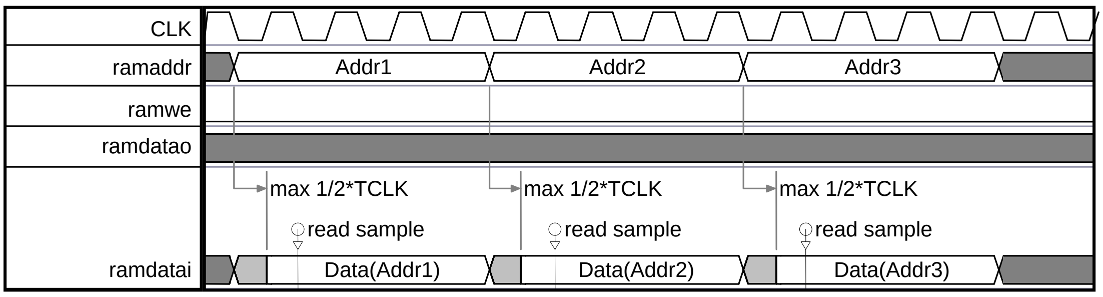


Figure 7. Data memory read cycle

Note:		
	TCLK	- time period of CLK signal
	Addr*n*	- address of memory cell
	Data(Addr*n*)	- data read from address Addr*n*
	read sample	- point at which data is read from bus into the internal register.

Data Memory Write Cycle

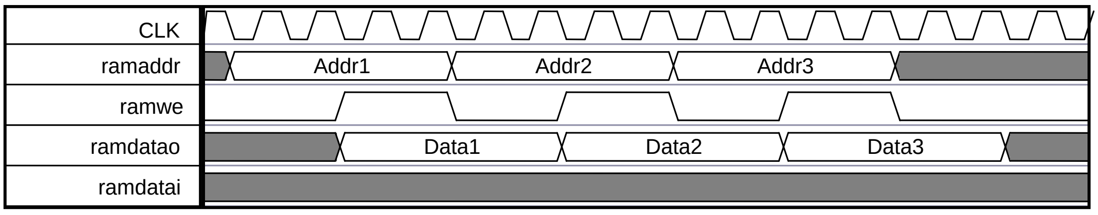


Figure 8. Data memory write cycle

Note:		
	TCLK	- time period of CLK signal
	Addr*n*	- address of Data memory cell
	Data*n*	- data to be written into address Addr*n*
	write sample	- point at which data is written from the bus into memory.

Data Memory Read and Write Cycle

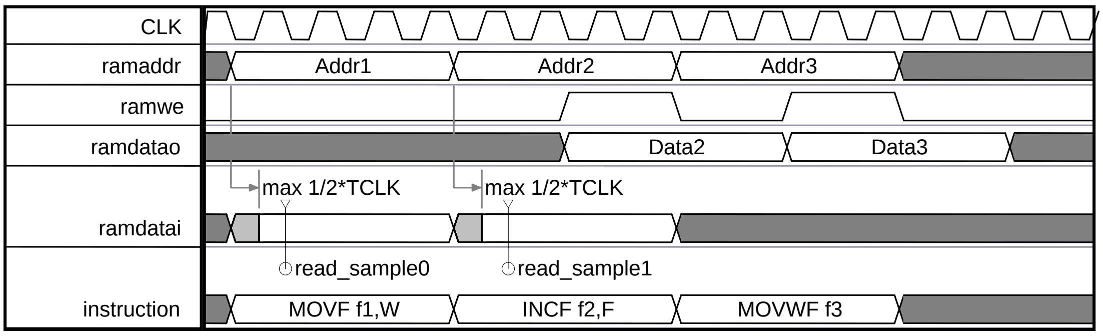


Figure 9. Data memory read and write cycle

Note:	TCLK	- time period of CLK signal
	Addrn	- address of Data memory cell
	Datan	- data to be written into address Addrn
	Data(Addrn)	- data read from address Addrn
	read sample	- point at which data is read from the bus into the internal register
	write sample	- point at which data is written from the bus into memory.

Special Function Registers

A map of the Special Function Registers is shown in Table 3. The Special Function Registers are registers used by the CPU and peripheral functions to control the operation of the device.

Table 3. Special Function Registers summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
N/A	TRIS	I/O control register (TRISA, TRISB, TRISC, TRISD[4], TRISE[4], TRISF[4])							
N/A	W	Work Register							
00h	INDF	Uses contents of FSR to address Data memory							
02h	PCL	Low order 8 bits of PC							
03h	STATUS[5]	PA2	PA1	PA0	TO	-	Z	DC	C
04h	FSR	Indirect Data memory address pointer							
05h	PORTA	RA7	RA6	RA5	RA4	RA3	RA2	RA1	RA0
06h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
07h	PORTC	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0
08h	PORTD[4]	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0
09h	PORTE[4]	RE7	RE6	RE5	RE4	RE3	RE2	RE1	RE0
0Ah	PORTF[4]	RF7	RF6	RF5	RF4	RF3	RF2	RF1	RF0

[4] TSK165C only

[5] In the TSK165A, bits 7:5 of the STATUS register are not used . In the TSK165B and TSK165C, bit 7 is not used.

Hardware Description

The structure of the TSK165x consists of:

- Control Unit
- Arithmetic Logic Unit
- Address Control Unit
- Ports unit
- Device Reset Timer
- Data memory RAM

Block Diagram

Figure 10 shows the hardware block diagram for the TSK165x.

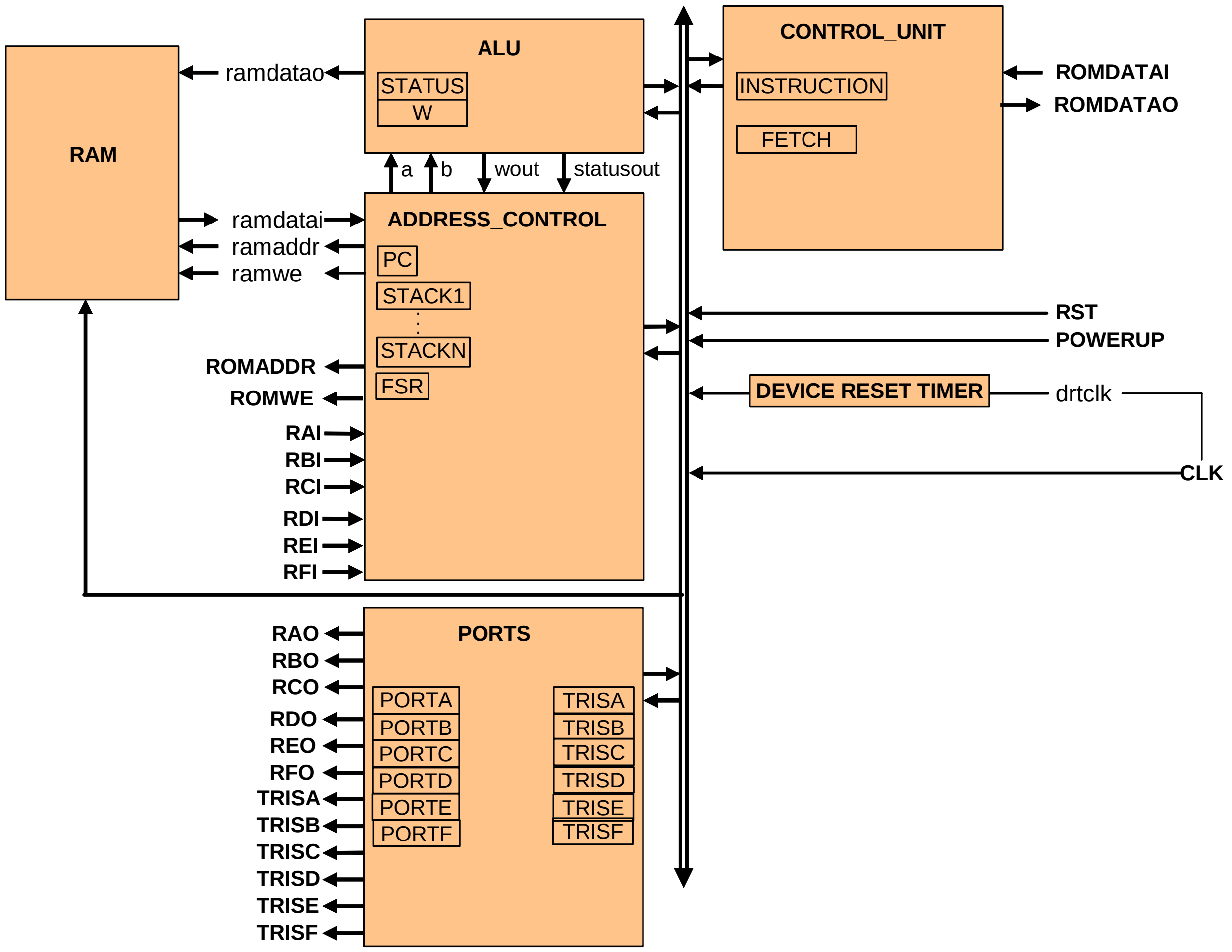


Figure 10. TSK165x Block diagram

Note that in the block diagram in Figure 10:

- The size of the RAM block depends on the TSK165x variant
  - 16 bytes for TSK165A
  - 64 bytes for TSK165B and TSK165C
- With respect to the Stack, N represents the depth of the stack
  - N=2 for TSK165A
  - N=8 for TSK165B and TSK165C
- The interface signals and registers for I/O ports D, E and F are available with the TSK165C only
- The Program memory interface signals ROMDATAO and ROMWE are available in the debug-enabled version of each core variant only.

Control Processor Unit

The Control Processor Unit (CPU) controls the fetching of instructions from Program memory. A fetched instruction is latched into the Instruction Register during the first clock cycle of the instruction cycle. This instruction is then decoded and executed during the remaining three clock cycles of the instruction cycle.

Arithmetic Logic Unit

The Arithmetic Logic Unit (ALU) is 8-bits wide and is capable of addition, subtraction, shift and logic operations. Unless otherwise mentioned, arithmetic operations are two's complement in nature. In two-operand instructions, typically one operand is the W (working) register. The other operand is either a file register or an immediate constant. The following sections describe the Arithmetic Logic Unit registers – W and STATUS - in greater detail.

Working Register (W))

The W register is an 8-bit working register used for ALU operations. It is not an addressable register.

STATUS Register (STATUS))

This register contains the arithmetic status of the ALU, the RESET status and the page preselect bits for Program memories larger than 512 bytes.

Table 4. The STATUS register flags

MSB							LSB
PA2	PA1	PA0	TO	-	Z	DC	C

Table 5. The STATUS register bit functions

Bit	Symbol	Function
7	PA2	Program memory page preselect bit 2
6	PA1	Program memory page preselect bit 1
5	PA0	Program memory page preselect bit 0
4	TO	Time-out bit
3	-	Not used
2	Z	Zero bit
1	DC	Digit carry bit
0	C	Carry bit

Note: In the TSK165A, bits 7:5 of the STATUS register are not used.

In the TSK165B and TSK165C, bit 7 is not used.

Program memory pages are selected based on the status of the page preselect bits. In the TSK165B and TSK165C, 2 bits are used to select the possible four pages (2K of memory space), as shown in Table 6.

Table 6. Program memory page location in the TSK165B and TSK165C

PA1/PA0	Page select	Location
00	Page 0	(000h – 1FFh)
01	Page 1	(200h – 3FFh)
10	Page 2	(400h – 5FFh)
11	Page 3	(600h – 7FFh)

Address Control Unit

The Address Control Unit interfaces to both Program and Data memory spaces and determines the source of an ALU operand. The unit consists of a Program Counter (PC), a Stack register (2-levels deep in the TSK165A; 8-levels deep in the TSK165B and TSK165C) and a File Select register (FSR).

Program Counter (PC))

As a program instruction is executed, the Program Counter (PC) will contain the address of the next program instruction to be executed. The PC is incremented by one at the start of the subsequent instruction cycle, unless an instruction changes the PC.

The low eight bits of the PC are mapped into the Data memory space as the PCL register.

For a GOTO instruction, bits 8:0 of the PC are provided by the GOTO instruction word. For the TSK165B and TSK165C (PC = 11 bits), the upper bits are provided by the page preselect bits in the STATUS register (PA1:PA0).

For a CALL instruction, or any instruction where the PCL is the destination, bits 7:0 of the PC again are provided by the instruction word. Bit 8 is always cleared and, in the case of the TSK165B and TSK165C, the upper bits of the PC are again provided by the page preselect bits.

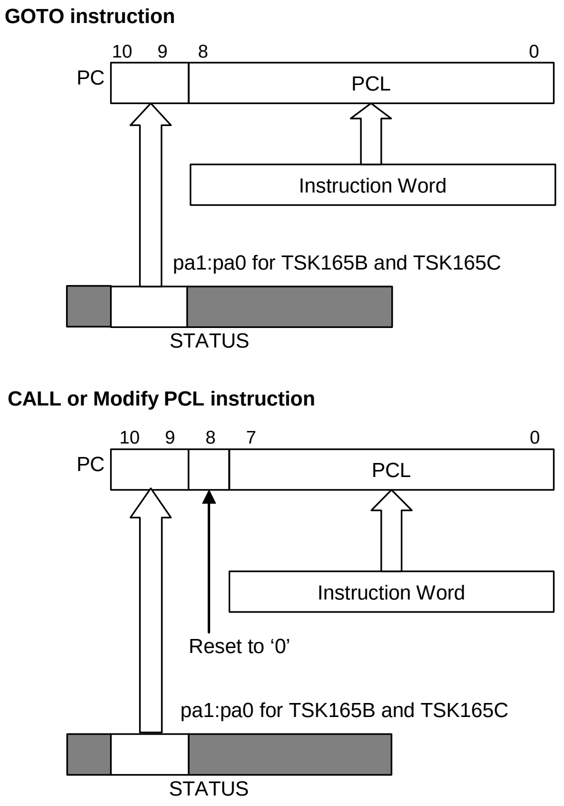


Figure 11. Loading of PC branch instruction.

Stack

A hardware push/pop stack is provided, whose width is the same as that of the Program Counter:

- TSK165A: 9-bit stack, 8 levels deep
- TSK165B, TSK165C: 11-bit stack, 8 levels deep

For all variants in the family:

- A CALL instruction will push the current value at each stack level down into the next stack level (e.g. stack1 into stack2, stack2 into stack3 and so on) and then push the current Program Counter value into stack1. Up to eight sequential CALL instructions can be executed, corresponding to the storage of the eight most recent return addresses.
- A RETLW instruction will pop the contents of stack1 into the Program Counter and then copy the contents in each level of the stack, up to the next level (e.g. stack8 into stack7, stack 7 into stack6 and so on). If more than eight sequential RETLW instructions are executed, the stack will be filled with the address previously stored in stack8.

For the RETLW instruction, the PC is loaded with the top of stack (tos) contents.

File Select Register (FSR))

This register is used for indirect data addressing. Addressing the INDF register actually addresses the Data memory whose address is contained in the FSR register. For example:

- Location 05h of Data memory contains the value 10h
- Location 06h of Data memory contains the value 0Ah
- Load the value 05h into the FSR register
- A read of the INDF register will return the value of 10h
- Increment the value of the FSR register by one (FSR=06h)
- A read of the INDF register now will return the value of 0Ah.

Reading INDF itself indirectly (FSR=0) will produce 00h. Writing to the INDF register indirectly results in a no-operation (NOP).

The bits 4:0 of the FSR are used to select Data memory addresses in the range 00h to 1Fh (32 bytes).

In the case of the TSK165B and TSK165C, where the Data memory space is larger than 32 bytes, bits 6:5 of the FSR are the bank select bits and are used to select the bank to be addressed (00 = Bank 0, 01 = Bank 1, 10 = Bank 2, 11 = Bank 3). Each bank addresses a further 32 bytes of Data memory space, although the top 16 bytes are identical to those in Bank 0. (Note that in the TSK165A, as the Data memory space is 32 bytes, no banking is required and so the upper bits of the FSR are not used).

Figure 12 shows the direct/indirect addressing for the TSK165A, while Figure 13 shows that for the TSK165B and TSK165C.

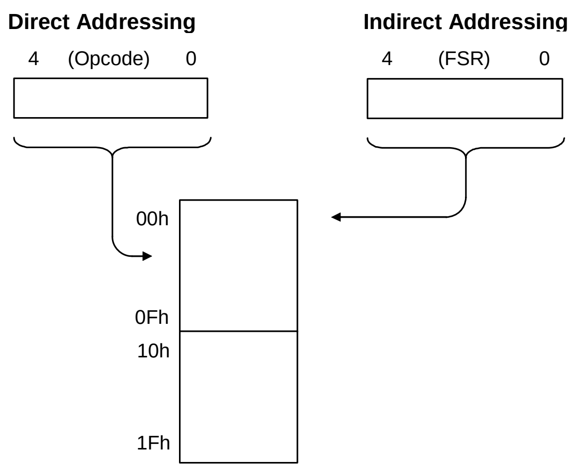


Figure 12. Direct/indirect addressing (TSK165A)

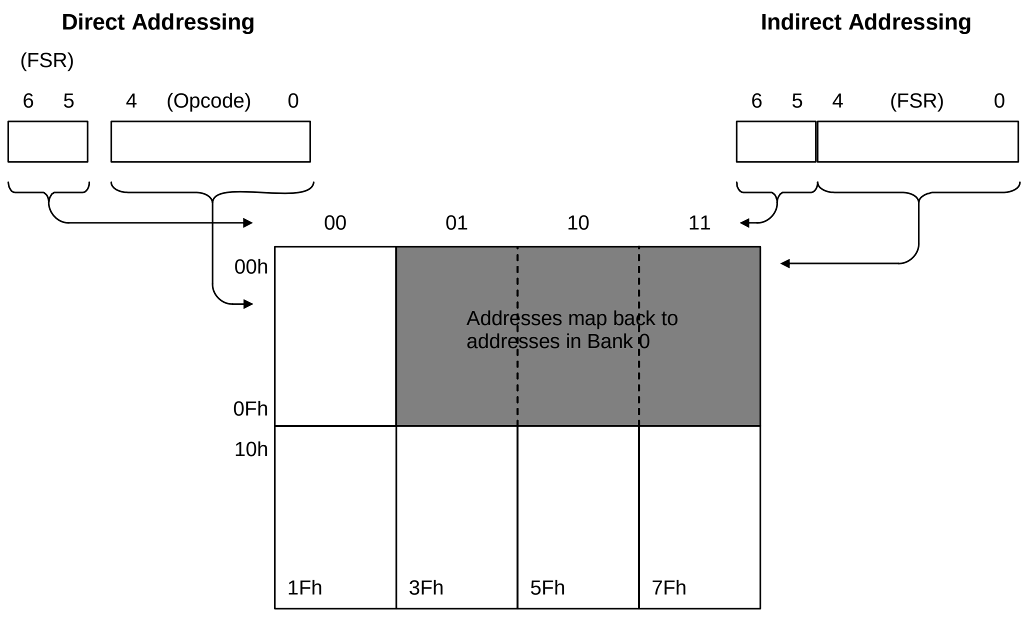


Figure 13. Direct/indirect addressing (TSK165B and TSK165C)

Ports Unit

The Ports Unit consists of port registers and TRIS registers. Read instructions always read the input ports (RAI, RBI and RCI). Write instructions always save the values in the port registers and drive data out through the output ports (RAO, RBO and RCO). The TRIS registers are output control registers.

Port Registers (Port A, Port B, Port C, Port D, Port E, Port F)

The number of port registers available depends on the particular core variant that is used:

TSK165A, TSK165B : three 8-bit port registers – Ports A, B and C

TSK165C : six 8-bit port registers – Ports A, B, C, D, E and F.

Each bit in each port can be configured, under program control, to be either an input or output. To simplify using these bi-directional ports, the schematic symbol includes a bus pin for each possible port configuration, allowing them to be wired independently (as shown in the example of Figure 14). Note that as the input mode is instantiated as a pin, any unused input ports must be tied appropriately. The simplest way to do this is to wire the input pin to the corresponding output pin.

Directional control is achieved through the corresponding TRIS registers for each port.

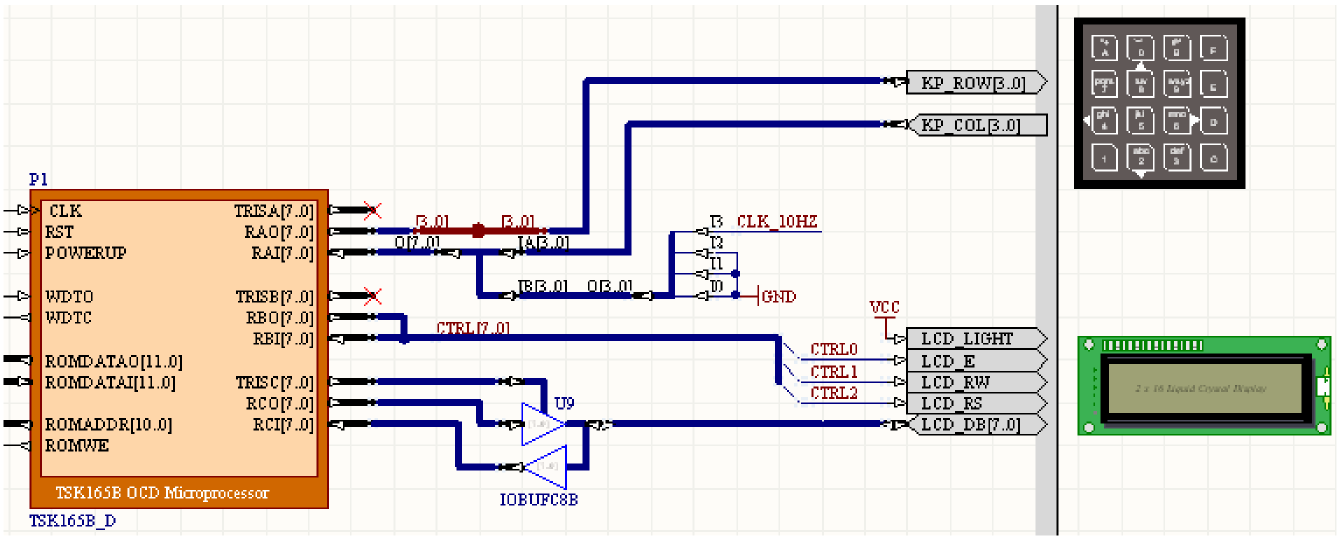


Figure 14. Port usage in a TSK165x microcontroller

TRIS Registers (TRISA, TRISB, TRISC, TRISD, TRISE, TRISF)

Each available port register in the device has a corresponding output driver control register:

TSK165A, TSK165B : TRISA – TRISC

TSK165C : TRISA – TRISF.

The output driver control registers are loaded with the contents of the W register by executing the TRIS f instruction. A ‘1’ from a TRIS register bit disables the corresponding output buffer and drive in hi-impedance mode. A ‘0’ from a TRIS register bit drives the contents output port (RAO, RBO, etc) on selected pins, enabling the output buffer.

Device Reset Timer

The Device Reset Timer is an 8-bit counter, incremented every falling edge of drtclk. This timer is for keeping the device in reset as long as the counter does not overflow. The source clock of this timer must be available to wake up and reset the system. This means that the drtclk clock must be running all the time. The source for that clock is a global clock and, in fact, the drtclk input is internally connected to the external system clock (CLK) input. The time-out on reset therefore depends on the frequency of CLK.

The DRT starts counting after any reset condition occurs.

Clock Domains

The TSK165x is synchronous and has the following clock domains:

- External system clock (CLK) for all registers of the microcontroller unit.
- Clock input for the Device Reset Timer counter (drtclk)

Note that drtclk is internally connected to CLK.

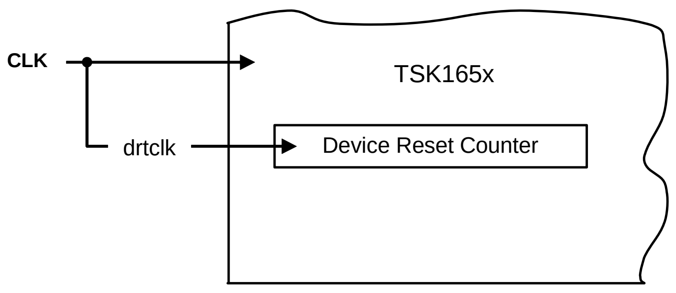


Figure 15. Clock domains

Resetting the Microcontroller

The TSK165x may be reset in one of the following ways:

- Hardware reset (RST)
- Reset on FPGA Initialization (POWERUP)
- Reset issued from an external Watchdog Timer peripheral (off-core)

Hardware Reset (RST)

This reset will occur if RST goes high. In this case, only the TRIS and PC registers, as well as the three high bits of the STATUS register are changed in accordance with their reset conditions. All other registers remain unchanged (see section *Reset Conditions*).

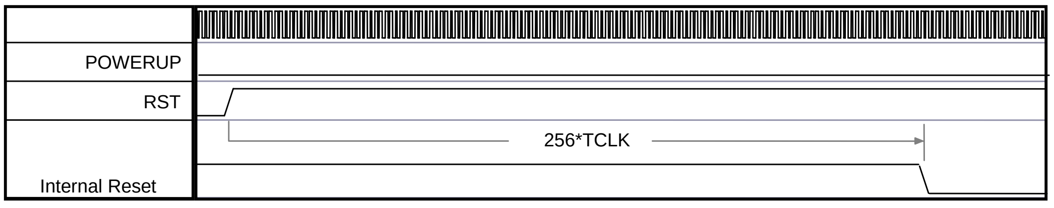


Figure 16. Reset timing on external reset

Note:	TCLK	- time period of 'CLK' signal
	CLK	- clock oscillator input
	Internal Reset	- internal signal generated based on an external reset condition
	POWERUP	- external reset input (Power On)
RST		- external reset input

Reset on FPGA Initialization (POWERUP)

This reset will occur whenever the physical FPGA device is powered up – either through download of a valid programming file or through the use of the POWERUP input (being taken high).

After the FPGA device has completed its initialization sequence, the processor will be held in the reset state for 256 cycles of the external system clock (CLK). This is shown in Figure 17 by the Internal Reset signal being held active (High). After this time, the processor will enter the normal 'Running' state, unless an additional external reset condition prevents it from doing so (e.g. POWERUP active, RST active, reset from an off-core Watchdog peripheral device).

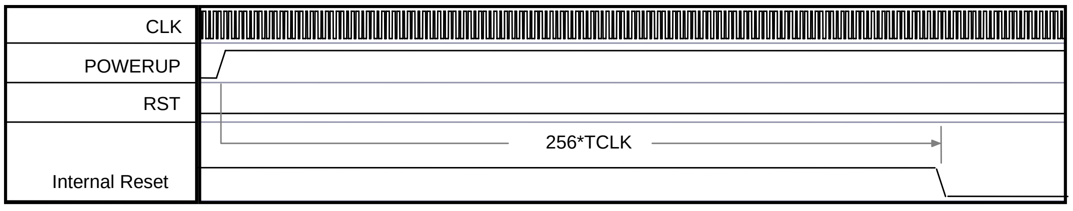


Figure 17. Reset timing on FPGA initialization.

Note:	TCLK	- time period of 'CLK' signal
	CLK	- clock oscillator input

Internal Reset	- internal signal generated based on an external reset condition
POWERUP	- external reset input (Power On)
RST	- external reset input.

Note: All registers will be in an unknown state (with the exception of the TRIS and PC registers and the five high bits of the STATUS register (see section *Reset Conditions*).

External Watchdog Timer Reset

This reset is produced by a Watchdog Timer (WDT) device. This device is a peripheral, placed externally to the TSK165x.

When the Watchdog Timer exceeds a defined count limit, an overflow signal is sent to the TSK165x, through the WDTO pin and the microcontroller is reset. The TO bit (STATUS<4>) is cleared upon reset.

The CLRWDT instruction is used to provide a clear signal to the Watchdog Timer, through the WDTC pin. The pin is taken high for a single clock cycle. During normal operation, this signal will be sent at regular intervals to prevent the Watchdog Timer from timing out and generating a device reset.

Reset Conditions

Table 7. Reset conditions for PCL and STATUS registers

Condition	**PCL** Addr: 02h	**STATUS** Addr: 03h
POWERUP	0000 0000	0001 -xxx
RST reset (normal operation)	0000 0000	000u -uuu
WDT reset (normal operation)	0000 0000	0000 -uuu

Legend: u = unchanged, x = unknown, - = not implemented (read as '0').

Table 8. Reset conditions for all registers

Register	**Address**	**POWERUP**	**RST or WDT Reset**
W	N/A	xxxx xxxx	uuuu uuuu
TRIS	N/A	1111 1111	1111 1111
INDF	00h	xxxx xxxx	uuuu uuuu
PCL	02h	0000 0000	0000 0000
STATUS	03h	0001 -xxx	000q -uuu
FSR	04h	1xxx xxxx	1uuu uuuu
PORTA	05h	xxxx xxxx	uuuu uuuu
PORTB	06h	xxxx xxxx	uuuu uuuu
PORTC	07h	xxxx xxxx	uuuu uuuu
PORTD[6]	08h	xxxx xxxx	uuuu uuuu
PORTE[6]	09h	xxxx xxxx	uuuu uuuu
PORTF[6]	0Ah	xxxx xxxx	uuuu uuuu

Legend: u = unchanged, x = unknown, - = not implemented (read as '0'), q = multiple values (see Table 7 for possible values).

[6] TSK165C only

Table 9. TO status after reset

TO	Reset was caused by
1	Power-up (POWERUP)
u	RST reset (normal operation)
0	WDT reset (normal operation)

Legend: u = unchanged.

Table 10. Events affecting TO status bit

Event	TO
Power up	1
External WDT Time-out	0
CLRWDT instruction	1

Legend: u = unchanged.

On-Chip Debugging

The debug-enabled versions of each of the core variants (TSK165A_D, TSK165B_D, TSK165C_D) provide the following set of additional functional features that facilitate real-time debugging of the microcontroller:

- Reset, Go, Halt processor control
- Single or multi-step debugging
- Read-write access for internal processor registers including SFRs and PC
- Read-write access for program memory and data memory
- Unlimited software breakpoints
- User can specify whether the peripheral's clocks are stopped when processor enters debug mode.

Adding Debug Functionality to a Standard Core Variant

For the TSK165A_D, TSK165B_D and TSK165C_D, the debug functionality is provided through the use of an On-Chip Debug System unit (OCDS). The simplified block diagram of Figure 18 shows the connection between this unit and the standard TSK165x core variant.

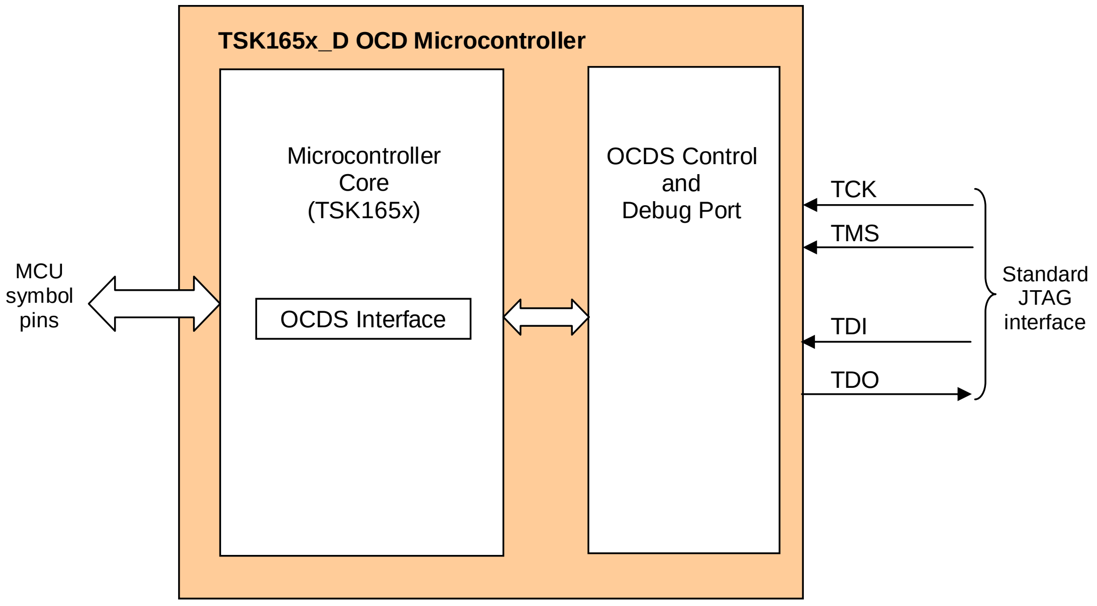

TSK165x_D OCD Microcontroller
Microcontroller Core (TSK165x)
OCDS Interface
OCDS Control and Debug Port
MCU symbol pins
TCK
TMS
TDI
TDO
Standard JTAG interface

Figure 18. Simplified TSK165x_D block diagram

The host computer is connected to the target core using the IEEE 1149.1 (JTAG) standard interface. This is the physical interface, providing connection to physical pins of the FPGA device in which the core has been embedded.

The Nexus 5001 standard is used as the protocol for communications between the host and all devices that are debug-enabled with respect to this protocol. This includes all OCD-version microcontrollers, as well as other Nexus-compliant devices such as frequency generators, logic analyzers, counters, etc.

All such devices are connected in a chain – the Soft Devices chain – which is determined when the design has been implemented within the target FPGA device and presents in the **Devices** view (Figure 19). It is not a physical chain, in the sense that you can see no external wiring – the connections required between the Nexus-enabled devices are made internal to the FPGA itself.

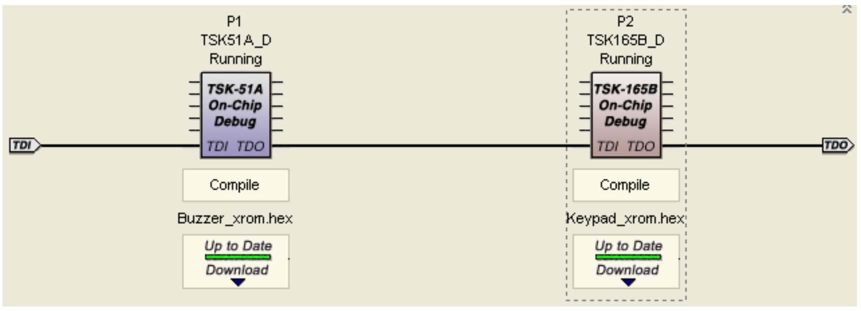


Figure 19. Nexus-enabled microcontrollers appearing in the Soft Devices chain

For microcontrollers such as the TSK165A_D, TSK165B_D and TSK165C_D, the Nexus protocol enables you to debug the core through communication with the OCDS Unit.

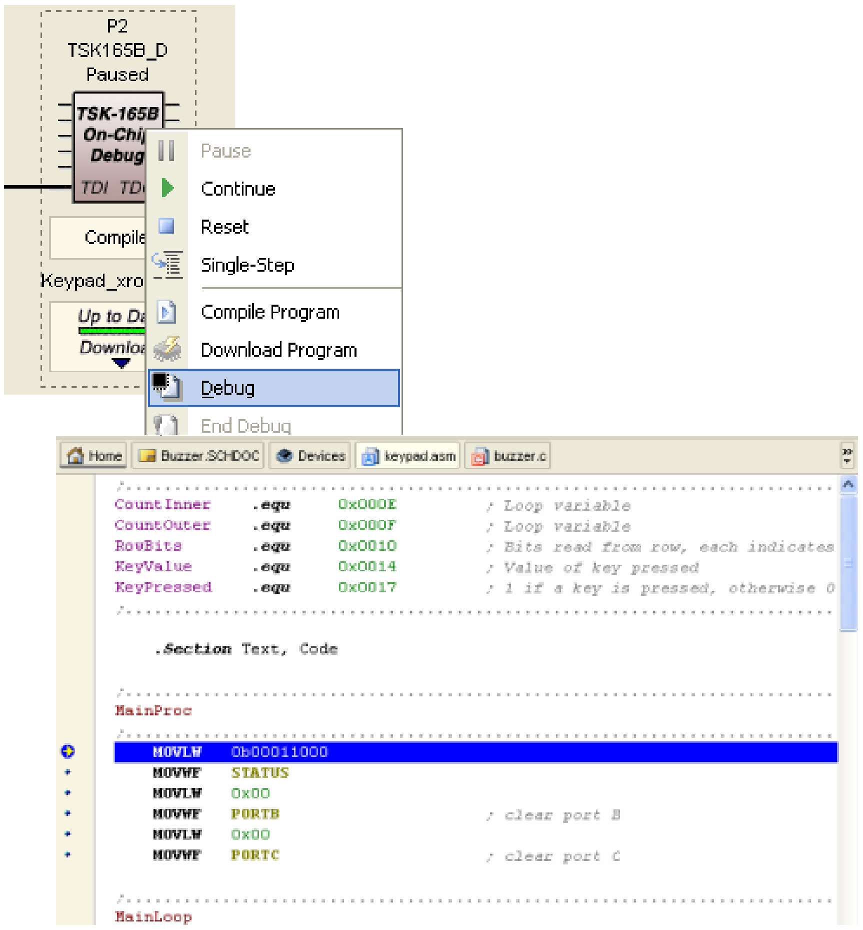
Accessing the Debug Environment

Debugging of the embedded code within an OCD-version microcontroller is carried out by starting a debug session. Prior to starting the session, you must ensure that the design, including one or more OCD-version microcontrollers and their respective embedded code, has been downloaded to the target physical FPGA device.

To start a debug session for the embedded code of a specific microcontroller in the design, simply right-click on the icon for that microcontroller, in the Soft Devices region of the view, and choose the **Debug** command from the pop-up menu that appears. Alternatively, click on the icon for the microcontroller (to focus it) and choose **Processors » Pn » Debug** from the main menus, where n corresponds to the number for the processor in the Soft Devices chain.

The embedded project for the software running in the processor will initially be recompiled and the debug session will commence. The relevant source code document (either Assembly or C) will be opened and the current execution point will be set to the first line of executable code (see Figure 20).

Note: You can have multiple debug sessions running simultaneously – one per embedded software project associated with a microcontroller in the Soft Devices chain.



```
;..........................................................................
CountInner      .equ      0x000E          ; Loop variable
CountOuter      .equ      0x000F          ; Loop variable
RowBits         .equ      0x0010          ; Bits read from row, each indicates
KeyValue        .equ      0x0014          ; Value of key pressed
KeyPressed      .equ      0x0017          ; 1 if a key is pressed, otherwise 0
;..........................................................................

    .Section Text, Code

;..........................................................................
MainProc
;..........................................................................
    MOVLW     0b00011000
    MOVWF     STATUS
    MOVLW     0x00
    MOVWF     PORTB                       ; clear port B
    MOVLW     0x00
    MOVWF     PORTC                       ; clear port C

;..........................................................................
MainLoop
```

Figure 20. Starting an embedded code debug session.

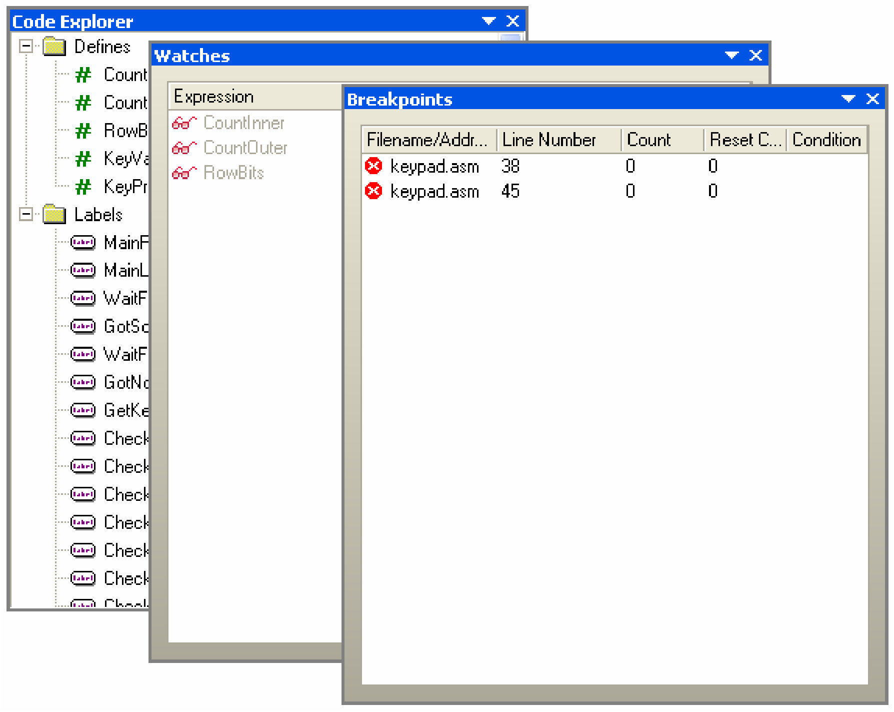
The debug environment offers the full suite of tools you would expect to see in order to efficiently debug the embedded code. These features include:

- Setting Breakpoints
- Adding Watches
- Stepping into and over at both the source (`*.C`) and instruction (`*.asm`) level
- Reset, Run and Halt code execution
- Run to cursor

All of these and other feature commands can be accessed from the **Debug** menu or the associated **Debug** toolbar.

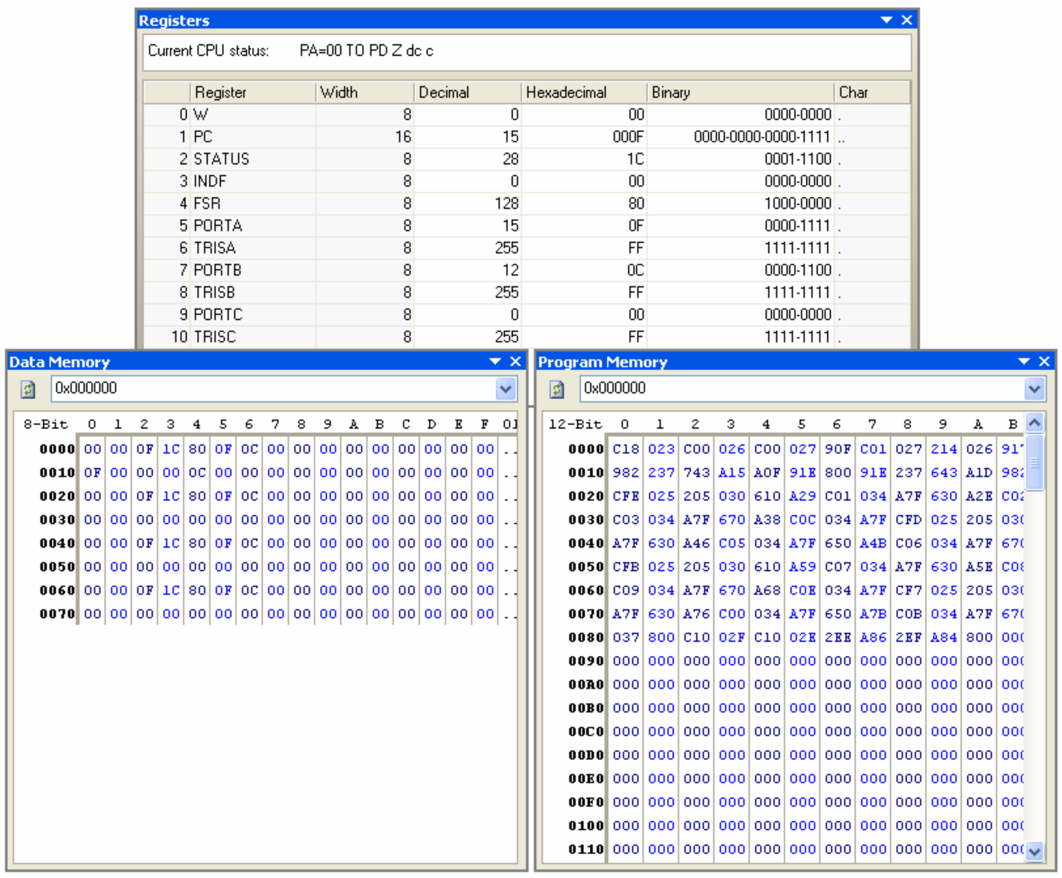
Various workspace panels are accessible in the debug environment, allowing you to view/control code-specific features, such as Breakpoints, Watches and Local variables, as well as information specific to the microcontroller in which the code is running, such as memory spaces and registers.

These panels can be accessed from the **View » Workspace Panels » Embedded** sub menu, or by clicking on the **Embedded** button at the bottom of the application window and choosing the required panel from the subsequent pop-up menu.



Filename/Addr...	Line Number	Count	Reset C...	Condition
keypad.asm	38	0	0	
keypad.asm	45	0	0	

Figure 21. Workspace panels offering code-specific information and controls



Current CPU status: PA=00 TO PD Z dc c

	Register	Width	Decimal	Hexadecimal	Binary	Char
0	W	8	0	00	0000-0000	.
1	PC	16	15	000F	0000-0000-0000-1111	..
2	STATUS	8	28	1C	0001-1100	.
3	INDF	8	0	00	0000-0000	.
4	FSR	8	128	80	1000-0000	.
5	PORTA	8	15	0F	0000-1111	.
6	TRISA	8	255	FF	1111-1111	.
7	PORTB	8	12	0C	0000-1100	.
8	TRISB	8	255	FF	1111-1111	.
9	PORTC	8	0	00	0000-0000	.
10	TRISC	8	255	FF	1111-1111	.

```
0x000000

8-Bit  0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  01
0000  00 00 0F 1C 80 0F 0C 00 00 00 00 00 00 00 00 00 ..
0010  0F 00 00 00 0C 00 00 00 00 00 00 00 00 00 00 00 ..
0020  00 00 0F 1C 80 0F 0C 00 00 00 00 00 00 00 00 00 ..
0030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
0040  00 00 0F 1C 80 0F 0C 00 00 00 00 00 00 00 00 00 ..
0050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
0060  00 00 0F 1C 80 0F 0C 00 00 00 00 00 00 00 00 00 ..
0070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
```

```
0x000000

12-Bit  0   1   2   3   4   5   6   7   8   9   A   B
0000  C18 023 C00 026 C00 027 90F C01 027 214 026 91
0010  982 237 743 A15 A0F 91E 800 91E 237 643 A1D 98
0020  CFE 025 205 030 610 A29 C01 034 A7F 630 A2E C0
0030  C03 034 A7F 670 A38 C0C 034 A7F CFD 025 205 03
0040  A7F 630 A46 C05 034 A7F 650 A4B C06 034 A7F 67
0050  CFB 025 205 030 610 A59 C07 034 A7F 630 A5E C0
0060  C09 034 A7F 670 A68 C0E 034 A7F CF7 025 205 03
0070  A7F 630 A76 C00 034 A7F 650 A7B C0B 034 A7F 67
0080  037 800 C10 02F C10 02E 2EE A86 2EF A84 800 00
0090  000 000 000 000 000 000 000 000 000 000 000 00
00A0  000 000 000 000 000 000 000 000 000 000 000 00
00B0  000 000 000 000 000 000 000 000 000 000 000 00
00C0  000 000 000 000 000 000 000 000 000 000 000 00
00D0  000 000 000 000 000 000 000 000 000 000 000 00
00E0  000 000 000 000 000 000 000 000 000 000 000 00
00F0  000 000 000 000 000 000 000 000 000 000 000 00
0100  000 000 000 000 000 000 000 000 000 000 000 00
0110  000 000 000 000 000 000 000 000 000 000 000 00
```

Figure 22. Workspace panels offering information specific to the parent processor.

Full-feature debugging is of course enjoyed at the source code level – from within the source code file itself. To a lesser extent, debugging can also be carried out from a dedicated debug panel for the processor. To access[7] this panel, first double-click on the icon representing the microcontroller to be debugged, in the **Soft Devices** region of the view. The *Instrument Rack – Soft Devices* panel will appear, with the chosen processor instrument added to the rack (Figure 23).

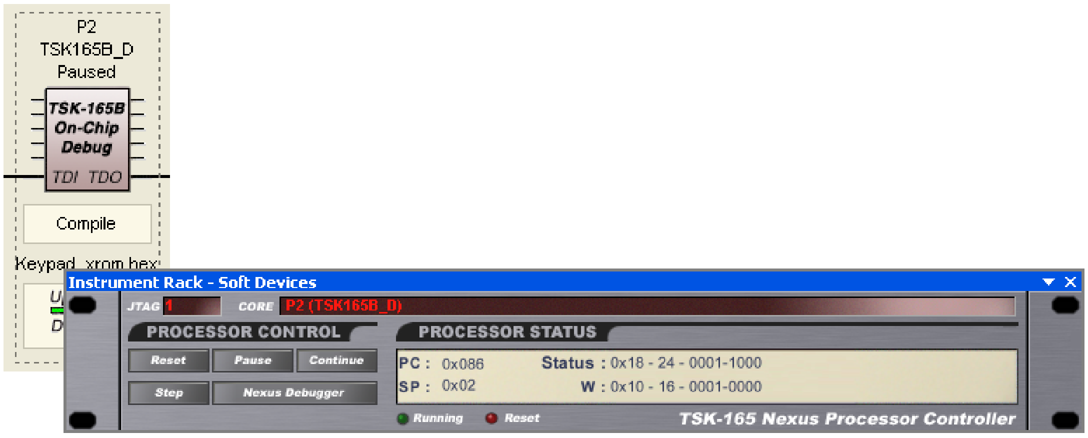


Figure 23. Accessing debug features from the microcontroller's instrument panel

Note: Each core microcontroller that you have included in the design will appear, when double-clicked, as an Instrument in the rack (along with any other Nexus-enabled devices).

The **Nexus Debugger** button provides access to the associated debug panel (Figure 24), which in turn allows you to interrogate and to a lighter extent control, debugging of the processor and its embedded code, notably with respect to the registers and memory.

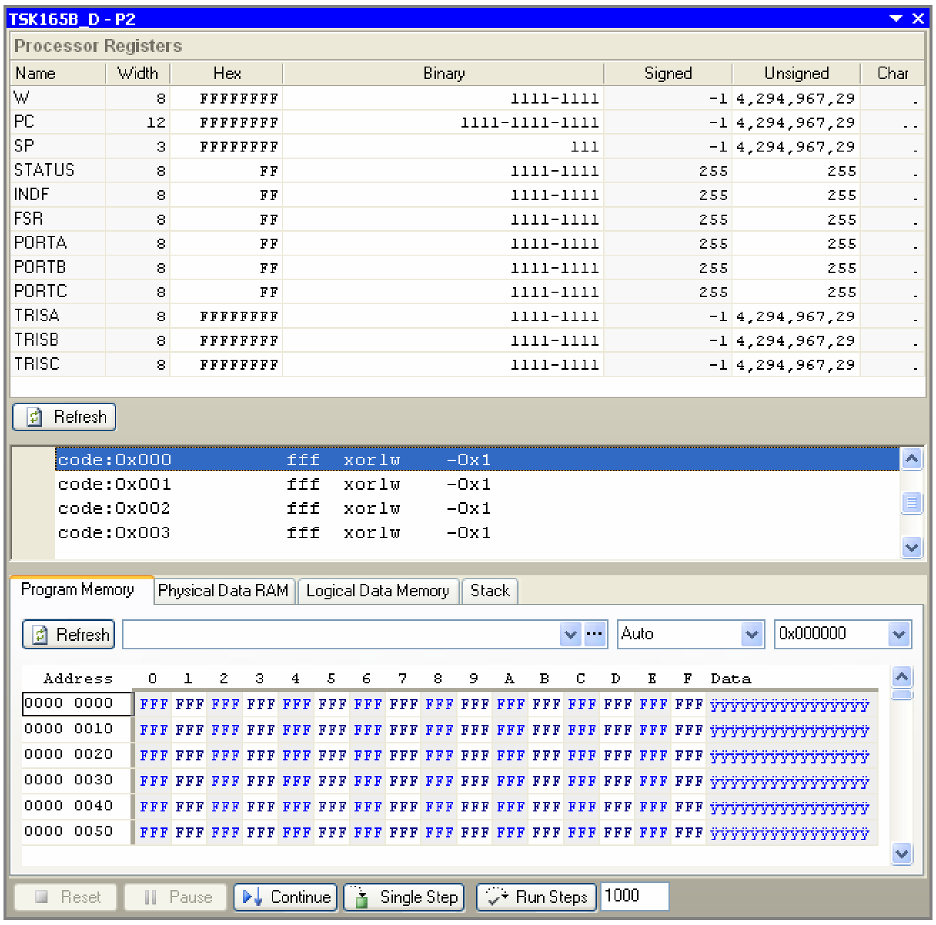
One key feature of the debug panel is that it enables you to specify (and therefore change) the embedded code (HEX file) that is downloaded to the microcontroller, quickly and efficiently.

For more information on the content and use of processor debug panels, press **F1** when the cursor is over one of these panels.

For further information regarding the use of the embedded tools for the TSK165x, see the *Using the TSK165x Embedded Tools* guide.

For comprehensive information with respect to the embedded tools available for the TSK165x, see the *TSK165x Embedded Tools Reference*.

[7] The debug panels for each of the debug-enabled microcontrollers are standard panels and, as such, can be readily accessed from the **View » Workspace Panels » Instruments** sub menu, or by clicking on the **Instruments** button at the bottom of the application window and choosing the required panel – for the processor you wish to debug – from the subsequent pop-up menu.



TSK165B_D - P2

Processor Registers

Name	Width	Hex	Binary	Signed	Unsigned	Char
W	8	FFFFFFFF	1111-1111	-1	4,294,967,29	.
PC	12	FFFFFFFF	1111-1111-1111	-1	4,294,967,29	..
SP	3	FFFFFFFF	111	-1	4,294,967,29	.
STATUS	8	FF	1111-1111	255	255	.
INDF	8	FF	1111-1111	255	255	.
FSR	8	FF	1111-1111	255	255	.
PORTA	8	FF	1111-1111	255	255	.
PORTB	8	FF	1111-1111	255	255	.
PORTC	8	FF	1111-1111	255	255	.
TRISA	8	FFFFFFFF	1111-1111	-1	4,294,967,29	.
TRISB	8	FFFFFFFF	1111-1111	-1	4,294,967,29	.
TRISC	8	FFFFFFFF	1111-1111	-1	4,294,967,29	.

```
code:0x000          fff  xorlw    -0x1
code:0x001          fff  xorlw    -0x1
code:0x002          fff  xorlw    -0x1
code:0x003          fff  xorlw    -0x1
```

Auto   0x000000

```
 Address    0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F  Data
0000 0000  FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0000 0010  FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0000 0020  FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0000 0030  FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0000 0040  FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0000 0050  FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
```

1000

Figure 24. Processor debugging using an associated processor debug panel

Instruction Set

All TSK165x instructions are binary code compatible. Each instruction comprises a 12-bit word divided into an Opcode, which specifies the instruction type, and one or more operands, which further specify the operation of the instruction.

Binary Opcode Instructions

Table 11 below summarizes the 12-bit Opcodes assigned to the various instructions in the set.

Table 11. Instruction Set in binary opcode

Mnemonic	**12-Bit Opcode**			**Mnemonic**	**12-Bit Opcode**		
	MSB		**LSB**		**MSB**		**LSB**
ADDWF	`0001`	`11df`	`ffff`	IORLW	`1101`	`kkkk`	`kkkk`
ANDLW	`1110`	`kkkk`	`kkkk`	IORWF	`0001`	`00df`	`ffff`
ANDWF	`0001`	`01df`	`ffff`	MOVF	`0010`	`00df`	`ffff`
BCF	`0100`	`bbbf`	`ffff`	MOVLW	`1100`	`kkkk`	`kkkk`
BSF	`0101`	`bbbf`	`ffff`	MOVWF	`0000`	`001f`	`ffff`
BTFSC	`0110`	`bbbf`	`ffff`	NOP	`0000`	`0000`	`0000`
BTFSS	`0111`	`bbbf`	`ffff`	RETLW	`1000`	`kkkk`	`kkkk`
CALL	`1001`	`kkkk`	`kkkk`	RLF	`0011`	`01df`	`ffff`
CLRF	`0000`	`011f`	`ffff`	RRF	`0011`	`00df`	`ffff`
CLRW	`0000`	`0100`	`0000`	SUBWF	`0000`	`10df`	`ffff`
CLRWDT	`0000`	`0000`	`0100`	SWAPF	`0011`	`10df`	`ffff`
COMF	`0010`	`01df`	`ffff`	TRIS	`0000`	`0000`	`0fff`
DECF	`0000`	`11df`	`ffff`	XORLW	`1111`	`kkkk`	`kkkk`
DECFSZ	`0010`	`11df`	`ffff`	XORWF	`0001`	`10df`	`ffff`
GOTO	`101k`	`kkkk`	`kkkk`				
INCF	`0010`	`10df`	`ffff`				
INCFSZ	`0011`	`11df`	`ffff`				

Opcode Field Descriptions

Table 12. Opcode field descriptions

Field	**Description**
f	Register file address (0x00 to 0x7F)
W	Working register (Accumulator)
b	Bit address within an 8-bit file register
k	Literal field, constant data or label
x	Don't care location (= 0 or 1)
d	Destination select: d = 0 (store result in W) d = 1 (store result in file register 'f') Default is d = 1

Field	Description
label	Label name
TOS	Top of Stack
PC	Program Counter
dest	Destination, either the W register or the specified register file location.

Instruction Format

Figure 25 shows the three general formats that instructions can have, depending on whether they are byte-oriented, bit-oriented or literal and control operations.

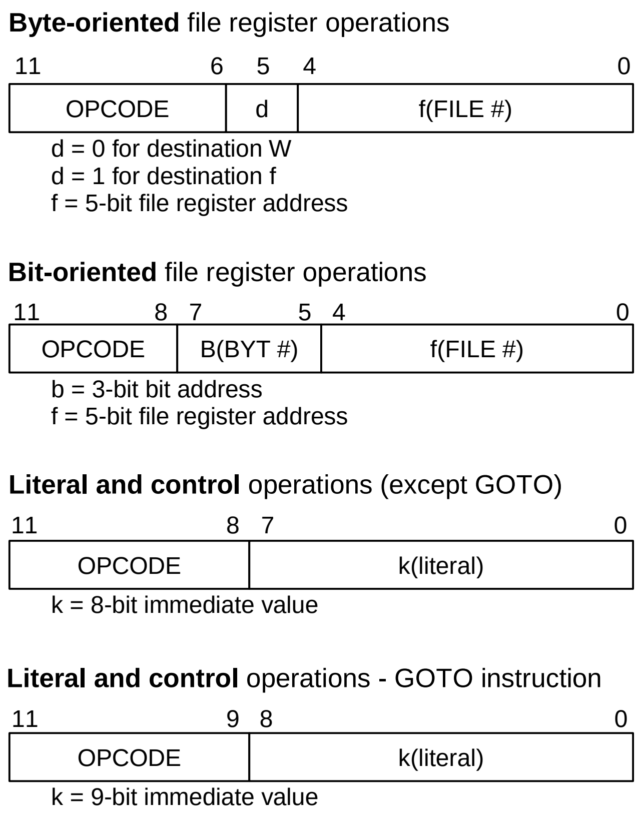

Byte-oriented file register operations
11
6
5
4
0
OPCODE
d
f(FILE #)
d = 0 for destination W
d = 1 for destination f
f = 5-bit file register address
Bit-oriented file register operations
11
8
7
5
4
0
OPCODE
B(BYT #)
f(FILE #)
b = 3-bit bit address
f = 5-bit file register address
Literal and control operations (except GOTO)
11
8
7
0
OPCODE
k(literal)
k = 8-bit immediate value
Literal and control operations - GOTO instruction
11
9
8
0
OPCODE
k(literal)
k = 9-bit immediate value

Figure 25. General format for instruction

Status Flag Modifications

Table 13 below shows the effect using each instruction has on the special function register STATUS. Only the lowest three bits of this register are affected – Bit 2 (C), Bit 1 (DC) and Bit 0 (Z).

Table 13. STATUS Flag modification (C, DC, Z)

Instruction	Flag			Instruction	Flag		
	C	DC	Z		C	DC	Z
ADDWF	X	X	X	IORLW			X
ANDLW			X	IORWF			X
ANDWF			X	MOVF			X
BCF				MOVLW			
BSF				MOVWF			
BTFSC				NOP			

Instruction	Flag			Instruction	Flag		
	C	DC	Z		C	DC	Z
BTFSS				RETLW			
CALL				RLF	X		
CLRF			1	RRF	X		
CLRW			1	SUBWF	X	X	X
CLRWDT				SWAPF			
COMF			X	TRIS			
DECF			X	XORLW			X
DECFSZ				XORWF			X
GOTO							
INCF			X				
INCFSZ							

Instruction Set – Functional Groupings

Table 14. Arithmetic operations

Mnemonic		**Description**	**Byte**	**Cycle**
ADDWF	f,d	Add W to f	1	1
DECF	f,d	Decrement f	1	1
INCF	f,d	Increment f	1	1
SUBWF	f,d	Subtract W from f	1	1

Table 15. Logic operations

Mnemonic		**Description**	**Byte**	**Cycle**
ANDLW	k	AND literal with W	1	1
ANDWF	f,d	AND W with f	1	1
CLRF	f	Clear f	1	1
CLRW	-	Clear W	1	1
CLRWDT	-	Clear Watchdog Timer	1	1
COMF	f,d	Complement f	1	1
IORLW	k	Inclusive OR literal with W	1	1
IORWF	f,d	Inclusive OR W with f	1	1
NOP	-	No operation	1	1
RLF	f,d	Rotate left f through Carry	1	1
RRF	f,d	Rotate right f through Carry	1	1
SWAPF	f,d	Swap f	1	1
XORLW	k	Exclusive OR literal with W	1	1
XORWF	f,d	Exclusive OR W with f	1	1

Table 16. Data transfer

Mnemonic		**Description**	**Byte**	**Cycle**
MOVF	f,d	Move f	1	1
MOVLW	k	Move literal to W	1	1
MOVWF	f	Move W to f	1	1
TRIS	f	Load TRIS register	1	1

Table 17. Boolean manipulation

Mnemonic		**Description**	**Byte**	**Cycle**
BCF	f,b	Bit clear f	1	1
BSF	f,b	Bit set f	1	1

Table 18. Program branches

Mnemonic		**Description**	**Byte**	**Cycle**
BTFSC	f,b	Bit Test f, Skip if Clear	1	1(2)
BTFSS	f,b	Bit Test f, Skip if Set	1	1(2)
CALL	k	Call subroutine	1	2
DECFSZ	f,d	Decrement f, Skip if zero	1	1(2)
GOTO	k	Unconditional branch	1	2
INCFSZ	f,d	Increment f, Skip if zero	1	1(2)
RETLW	k	Return, place literal in W	1	2

Instruction Set – Detailed Reference

ADDWF f,d

Description: Add the contents of the W register and register ‘f’. If ‘d’ is 0, the result is stored in the W register. If ‘d’ is 1, the result is stored back in register ‘f’.

Operation: (W) + (f) $\rightarrow$ (dest)

Words: 1

Status Affected: C, DC, Z

Encoding:

0	0	0	1	1	1	d	f	f	f	f	f

ANDLW k

Description: The contents of the W register are AND’ed with the eight-bit literal ‘k’. The result is placed in the W register.

Operation: (W) AND (k) $\rightarrow$ (W)

Words: 1

Status Affected: Z

Encoding:

1	1	1	0	k	k	k	k	k	k	k	k

ANDWF f,d

Description: The contents of the W register are AND'ed with register 'f'. If 'd' is 0, the result is stored in the W register. If 'd' is 1, the result is stored back in register 'f'.

Operation: (W) AND (f) $\rightarrow$ (dest)

Words: 1

Status Affected: Z

Encoding:

0	0	0	1	0	1	d	f	f	f	f	f

BCF f,b

Description: Bit 'b' in register 'f' is cleared.

Operation: 0 $\rightarrow$ (f<b>)

Words: 1

Status Affected: None

Encoding:

0	1	0	0	b	b	b	f	f	f	f	f

BSF f,b

Description: Bit 'b' in register 'f' is set.

Operation: 1 $\rightarrow$ (f<b>)

Words: 1

Status Affected: None

Encoding:

0	1	0	1	b	b	b	f	f	f	f	f

BTFSC f,b

Description: If bit 'b' in register 'f' is 0 then the next instruction is skipped.

Operation: skip if (f<b>)=0

Words: 1

Status Affected: None

Encoding:

0	1	1	0	b	b	b	f	f	f	f	f

BTFSS f,b

Description: If bit 'b' in register 'f' is 1 then the next instruction is skipped.

Operation: skip if (f<b>)=1

Words: 1

Status Affected: None

Encoding:

0	1	1	1	b	b	b	f	f	f	f	f

CALL k

Description: Subroutine call. First, return address (PC+1) is pushed onto the stack. The eight bit immediate address is loaded into PC bits <7:0>.

For the TSK165B and TSK165C (PC 11 bits), the upper bits are loaded from STATUS<6:5>.

PC<8> is always cleared.

CALL is a two cycle instruction.

Operation: (PC) $\rightarrow$ Top of STACK

k $\rightarrow$ PC<7:0>

(STATUS<6:5>) $\rightarrow$ PC<10:9> - (in the case of the TSK165B and TSK165C)

0 $\rightarrow$ PC<8>

Words: 1

Status Affected: None

Encoding:

1	0	0	1	k	k	k	k	k	k	k	k

CLRF f

Description: The contents of register ‘f’ are cleared and the Zero bit (Z) bit is set.

Operation: 00h $\rightarrow$ (f)

1 $\rightarrow$ Z

Words: 1

Status Affected: Z

Encoding:

0	0	0	0	0	1	1	f	f	f	f	f

CLRW

Description: The W register is cleared. Zero bit (Z) is set.

Operation: 00h $\rightarrow$ (W)

1 $\rightarrow$ Z

Words: 1

Status Affected: Z

Encoding:

0	0	0	0	0	1	0	0	0	0	0	0

CLRWDT

Description: This instruction resets the Watchdog Timer (WDT). The Watchdog Timer is an independently placed peripheral component, external to the TSK165x. Using this instruction essentially takes the WDTC pin high for a single clock cycle, providing the clear impulse signal to the WDT. The STATUS bit TO is set.

Operation: 1 $\rightarrow$ TO

Words: 1

Status Affected: TO

Encoding:

0	0	0	0	0	0	0	0	0	1	0	0

COMF f,d

Description: The contents of register ‘f’ are complemented. If ‘d’ is 0, the result is stored in the W register. If ‘d’ is 1, the result is stored back in register ‘f’.

Operation: NOT (f) → (dest)

Words: 1

Status Affected: Z

Encoding:

0	0	1	0	0	1	d	f	f	f	f	f

DECF f,d

Description: Decrement register ‘f’. If ‘d’ is 0, the result is stored in the W register. If ‘d’ is 1, the result is stored back in register ‘f’.

Operation: (f) - 1 → (dest)

Words: 1

Status Affected: Z

Encoding:

0	0	0	0	1	1	d	f	f	f	f	f

DECFSZ f,d

Description: The contents of register ‘f’ are decremented. If ‘d’ is 0, the result is placed in the W register. If ‘d’ is 1, the result is placed back in register ‘f’. If the result is 0, the next instruction, which is already fetched, is discarded and an NOP is executed instead, making it a two cycle instruction.

Operation: (f) - 1 → (dest) skip if result = 0

Words: 1

Status Affected: None

Encoding:

0	0	1	0	1	1	d	f	f	f	f	f

GOTO k

Description: GOTO is an unconditional branch. The 9-bit immediate value is loaded into PC bits <8:0>.

For the TSK165B and TSK165C (PC 11 bits), the upper bits are loaded from STATUS<6:5>.

GOTO is a two cycle instruction.

Operation: k → PC<8:0>

(STATUS<6:5>) → PC<10:9> (in the case of the TSK165B and TSK165C)

Words: 1

Status Affected: None

Encoding:

1	0	1	k	k	k	k	k	k	k	k	k

INCF f,d

Description: The contents of register ‘f’ are incremented. If ‘d’ is 0, the result is placed in the W register. If ‘d’ is 1, the result is placed back in register ‘f’.

Operation: (f) + 1 → (dest)

Words: 1

Status Affected: Z

Encoding:

0	0	1	0	1	0	d	f	f	f	f	f

INCFSZ f,d

Description: The contents of register ‘f’ are incremented. If ‘d’ is 0, the result is placed in the W register. If ‘d’ is 1, the result is placed back in register ‘f’. If the result is 0, then the next instruction, which is already fetched, is discarded and an NOP is executed instead, making it a two cycle instruction.

Operation: $(f) + 1 \rightarrow (dest)$, skip if result = 0

Words: 1

Status Affected: None

Encoding:

0	0	1	1	1	1	d	f	f	f	f	f

IORLW k

Description: The contents of the W register are OR’ed with the eight bit literal ‘k’. The result is placed in the W register.

Operation: (W) OR $(k) \rightarrow (W)$

Words: 1

Status Affected: Z

Encoding:

1	1	0	1	k	k	k	k	k	k	k	k

IORWF f,d

Description: Inclusive OR the W register with register ‘f’. If ‘d’ is 0, the result is placed in the W register. If ‘d’ is 1, the result is placed back in register ‘f’.

Operation: (W) OR $(f) \rightarrow (dest)$

Words: 1

Status Affected: Z

Encoding:

0	0	0	1	0	0	d	f	f	f	f	f

MOVF f,d

Description: The contents of register ‘f’ are moved to destination ‘d’. If ‘d’ is 0, the destination is the W register. If ‘d’ is 1, the destination is the file register ‘f’.

Operation: $(f) \rightarrow (dest)$

Words: 1

Status Affected: Z

Encoding:

0	0	1	0	0	0	d	f	f	f	f	f

MOVLW k

Description: The eight bit literal ‘k’ is loaded into the W register. The don’t cares will assemble as 0s.

Operation: $k \rightarrow (W)$

Words: 1

Status Affected: None

Encoding:

1	1	0	0	k	k	k	k	k	k	k	k

MOVWF f

Description: Move data from the W register to register ‘f’.

Operation: (W) → (f)

Words: 1

Status Affected: None

Encoding:

0	0	0	0	0	0	1	f	f	f	f	f

NOP

Description: No operation.

Operation: No operation.

Words: 1

Status Affected: None

Encoding:

0	0	0	0	0	0	0	0	0	0	0	0

RETLW k

Description: The W register is loaded with the eight bit literal ‘k’. The program counter is loaded from the top of stack (the return address). This is a two cycle instruction.

Operation: k → (W)

TOS → PC

Words: 1

Status Affected: None

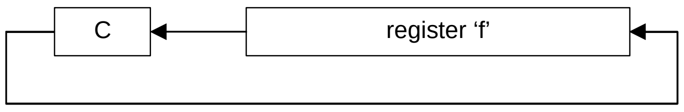
Encoding:

1	0	0	0	k	k	k	k	k	k	k	k

RLF f,d

Description: The contents of register ‘f’ are rotated one bit to the left through the Carry Flag. If ‘d’ is 0, the result is placed in the W register. If ‘d’ is 1, the result is stored back in register ‘f’.

Operation: See description below.



Words: 1

Status Affected: C

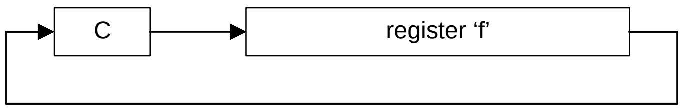
Encoding:

0	0	1	1	0	1	d	f	f	f	f	f

RRF f,d

Description: The contents of register ‘f’ are rotated one bit to the right through the Carry Flag. If ‘d’ is 0, the result is placed in the W register. If ‘d’ is 1, the result is stored back in register ‘f’.

Operation: See description below.



Words: 1

Status Affected: C

Encoding:

0	0	1	1	0	0	d	f	f	f	f	f

SUBWF f,d

Description: Subtract (2's complement method) the W register from register 'f'. If 'd' is 0, the result is stored in the W register. If 'd' is 1, the result is stored back in register 'f'.

Operation: (f) – (W) $\rightarrow$ (dest)

Words: 1

Status Affected: C, DC, Z

Encoding:

0	0	0	0	1	0	d	f	f	f	f	f

SWAPF f,d

Description: The upper and lower nibbles of register 'f' are exchanged. If 'd' is 0, the result is placed in the W register. If 'd' is 1, the result is placed in register 'f'.

Operation: (f<3:0>) $\rightarrow$ (dest<7:4>)

(f<7:4>) $\rightarrow$ (dest<3:0>)

Words: 1

Status Affected: None

Encoding:

0	0	1	1	1	0	d	f	f	f	f	f

TRIS f

Description: TRIS register 'f' (f = 5, 6 or 7) is loaded with the contents of the W register.

Operation: (W) $\rightarrow$ TRIS register f

Words: 1

Status Affected: None

Encoding:

0	0	0	0	0	0	0	0	0	f	f	f

XORLW k

Description: The contents of the W register are XOR'ed with the eight bit literal 'k'. The result is placed in the W register.

Operation: (W) XOR k $\rightarrow$ (W)

Words: 1

Status Affected: Z

Encoding:

1	1	1	1	k	k	k	k	k	k	k	k

XORWF f,d

Description: Exclusive OR the contents of the W register with register 'f'. If 'd' is 0, the result is stored in the W register. If 'd' is 1, the result is stored back in register 'f'.

Operation: (W) XOR (f) $\rightarrow$ (dest)

Words: 1

Status Affected: Z

Encoding:

0	0	0	1	1	0	d	f	f	f	f	f

Instruction Timing

The instruction fetch and execute are pipelined such that a fetch takes one instruction cycle while decode and execute takes another instruction cycle. However, due to the pipelining, each instruction effectively executes in one cycle. If an instruction causes the Program Counter to change, then two cycles are required to complete the instruction (Table 19).

Table 19. Instruction pipeline flow

MOVLW 55H	Fetch 1	Execute 1				
MOVWF PORTB		Fetch 2	Execute 2			
CALL SUB1			Fetch 3	Execute 3		
BSF PORTA,BIT3				Fetch 4	Flush	
Instr a addr SUB1					Fetch SUB1	Execute SUB1
						Fetch SUB1+1

All instructions are single cycle, except for any program branches. These take two cycles since the fetch instruction is “flushed” from the pipeline while the new instruction is being fetched and then executed.

Revision History

Date	Version No.	Revision
20-Jan-2004	1.0	New product release
22-Oct-2004	1.1	Modifications to example design images and images used in the On-Chip Debugging section. Addition of Nexus Debugger panel.
08-Feb-2005	1.2	Modifications to debug panel information in On-Chip Debugging section.
09-May-2005	1.3	Updated for SP4
12-Dec-2005	1.4	Path references updated for Altium Designer 6
13-Mar-2008	2.0	Updated for Altium Designer Summer 08