Left: E:/Programas/Arduino1.8.8/hardware/teensy/avr/cores/teensy3/serial2.c.original
Right: E:/Programas/Arduino1.8.8/hardware/teensy/avr/cores/teensy3/serial2.c

Left: Wed Dec 26 06:32:05 2018	Right: Wed Sep 11 21:30:40 2019
Line 61	**Line 61**
static volatile BUFTYPE tx_buffer[SERIAL2_TX_BUFFER_SIZE];	static volatile BUFTYPE tx_buffer[SERIAL2_TX_BUFFER_SIZE];
static volatile BUFTYPE rx_buffer[SERIAL2_RX_BUFFER_SIZE];	static volatile BUFTYPE rx_buffer[SERIAL2_RX_BUFFER_SIZE];
static volatile uint8_t transmitting = 0;	static volatile uint8_t transmitting = 0;
	static volatile uint8_t queueing_tx_data = 0; // PedroR
	#define TX_BUFFER_EMPTY (tx_buffer_head == tx_buffer_tail) // PedroR
#if defined(KINETISK)	#if defined(KINETISK)
static volatile uint8_t *transmit_pin=NULL;	static volatile uint8_t *transmit_pin=NULL;
	static volatile uint8_t *debug_pin=NULL; // PedroR
#define transmit_assert() *transmit_pin = 1	#define transmit_assert() *transmit_pin = 1
#define transmit_deassert() *transmit_pin = 0	#define transmit_deassert() *transmit_pin = 0
static volatile uint8_t *rts_pin=NULL;	static volatile uint8_t *rts_pin=NULL;
Line 117	**Line 120**
tx_buffer_head = 0;	tx_buffer_head = 0;
tx_buffer_tail = 0;	tx_buffer_tail = 0;
transmitting = 0;	transmitting = 0;
	queueing_tx_data = 0; // PedroR
#if defined(KINETISK)	#if defined(KINETISK)
switch (rx_pin_num) {	switch (rx_pin_num) {
case 9: CORE_PIN9_CONFIG = PORT_PCR_PE | PORT_PCR_PS | PORT_PCR_PFE | PORT_PCR_MUX(3); break;	case 9: CORE_PIN9_CONFIG = PORT_PCR_PE | PORT_PCR_PS | PORT_PCR_PFE | PORT_PCR_MUX(3); break;
Line 134	**Line 139**
case 58: CORE_PIN58_CONFIG = PORT_PCR_DSE | PORT_PCR_SRE | PORT_PCR_MUX(3); break;	case 58: CORE_PIN58_CONFIG = PORT_PCR_DSE | PORT_PCR_SRE | PORT_PCR_MUX(3); break;
#endif	#endif
}	}
	/** Pedro R: use pin 6 for debug */
	pinMode(6, OUTPUT);
	digitalWrite(6, LOW);
	debug_pin = portOutputRegister(6);
	/****/
#elif defined(KINETISL)	#elif defined(KINETISL)
CORE_PIN9_CONFIG = PORT_PCR_PE | PORT_PCR_PS | PORT_PCR_PFE | PORT_PCR_MUX(3);	CORE_PIN9_CONFIG = PORT_PCR_PE | PORT_PCR_PS | PORT_PCR_PFE | PORT_PCR_MUX(3);
CORE_PIN10_CONFIG = PORT_PCR_DSE | PORT_PCR_SRE | PORT_PCR_MUX(3);	CORE_PIN10_CONFIG = PORT_PCR_DSE | PORT_PCR_SRE | PORT_PCR_MUX(3);
Line 352	**Line 365**
uint32_t head, n;	uint32_t head, n;
if (!(SIM_SCGC4 & SIM_SCGC4_UART1)) return;	if (!(SIM_SCGC4 & SIM_SCGC4_UART1)) return;
	// PedroR: protect this code block
	queueing_tx_data = 1;
if (transmit_pin) transmit_assert();	if (transmit_pin) transmit_assert();
head = tx_buffer_head;	head = tx_buffer_head;
if (++head >= SERIAL2_TX_BUFFER_SIZE) head = 0;	if (++head >= SERIAL2_TX_BUFFER_SIZE) head = 0;
while (tx_buffer_tail == head) {	while (tx_buffer_tail == head) {
int priority = nvic_execution_priority();	int priority = nvic_execution_priority();
if (priority <= IRQ_PRIORITY) {	if (priority <= IRQ_PRIORITY) {
	// PedroR: added bc it is also in serial2_Write but wasn't here
	queueing_tx_data = 0; // Pedro R: need to be 0 so that C2_TX_ACTIVE fires the necessary interrupts properly
	UART1_C2 = C2_TX_ACTIVE;
	// end PedroR added
if ((UART1_S1 & UART_S1_TDRE)) {	if ((UART1_S1 & UART_S1_TDRE)) {
uint32_t tail = tx_buffer_tail;	uint32_t tail = tx_buffer_tail;
if (++tail >= SERIAL2_TX_BUFFER_SIZE) tail = 0;	if (++tail >= SERIAL2_TX_BUFFER_SIZE) tail = 0;
Line 373	**Line 394**
tx_buffer[head] = c;	tx_buffer[head] = c;
transmitting = 1;	transmitting = 1;
tx_buffer_head = head;	tx_buffer_head = head;
	queueing_tx_data = 0;
	// PedroR: end protecting code block
	// An interrupt can still occur between clearing the flag and setting UART_C2; therefore we will also check for
	// (!TX_BUFFER_EMPTY) in the interrupts, in addition to the queueing_tx_data flag
UART1_C2 = C2_TX_ACTIVE;	UART1_C2 = C2_TX_ACTIVE;
}	}
#ifdef HAS_KINETISK_UART1_FIFO	#ifdef HAS_KINETISK_UART1_FIFO
Line 384	**Line 413**
uint32_t head, n;	uint32_t head, n;
if (!(SIM_SCGC4 & SIM_SCGC4_UART1)) return;	if (!(SIM_SCGC4 & SIM_SCGC4_UART1)) return;
	// PedroR: protect this code block
	queueing_tx_data = 1;
if (transmit_pin) transmit_assert();	if (transmit_pin) transmit_assert();
while (p < end) {	while (p < end) {
head = tx_buffer_head;	head = tx_buffer_head;
if (++head >= SERIAL2_TX_BUFFER_SIZE) head = 0;	if (++head >= SERIAL2_TX_BUFFER_SIZE) head = 0;
if (tx_buffer_tail == head) {	if (tx_buffer_tail == head) {
	queueing_tx_data = 0; // Pedro R: need to be 0 so that C2_TX_ACTIVE fires the necessary interrupts properly
UART1_C2 = C2_TX_ACTIVE;	UART1_C2 = C2_TX_ACTIVE;
do {	do {
int priority = nvic_execution_priority();	int priority = nvic_execution_priority();
Line 405	**Line 439**
yield();	yield();
}	}
} while (tx_buffer_tail == head);	} while (tx_buffer_tail == head);
}	
	queueing_tx_data = 1; // Pedro R: restore protection to code block
tx_buffer[head] = *p++;	tx_buffer[head] = *p++;
transmitting = 1;	transmitting = 1;
tx_buffer_head = head;	tx_buffer_head = head;
}	}
	queueing_tx_data = 0;
	// PedroR: end protecting code block
	// An interrupt can still occur between clearing the flag and setting UART_C2; therefore we will also check for
	// (!TX_BUFFER_EMPTY) in the interrupts, in addition to the queueing_tx_data flag
UART1_C2 = C2_TX_ACTIVE;	UART1_C2 = C2_TX_ACTIVE;
}	}
#else	#else
Line 553	**Line 596**
}	}
}	}
c = UART1_C2;	c = UART1_C2;
if ((c & UART_C2_TIE) && (UART1_S1 & UART_S1_TDRE)) {	if ((c & UART_C2_TIE) && (UART1_S1 & UART_S1_TDRE)) {
head = tx_buffer_head;	head = tx_buffer_head;
tail = tx_buffer_tail;	tail = tx_buffer_tail;
do {	do {
Line 565	**Line 608**
UART1_D = n;	UART1_D = n;
} while (UART1_TCFIFO < 8);	} while (UART1_TCFIFO < 8);
tx_buffer_tail = tail;	tx_buffer_tail = tail;
if (UART1_S1 & UART_S1_TDRE) UART1_C2 = C2_TX_COMPLETING;	if (UART1_S1 & UART_S1_TDRE) {
	// PedroR
	if (queueing_tx_data) {
	UART1_C2 = C2_TX_INACTIVE; // the code block protected by queueing_tx_data will address UART_C2 and re activate it
	} else if (!TX_BUFFER_EMPTY) {
	UART1_C2 = C2_TX_ACTIVE; // force interrupts to continue firing
	} else {
	UART1_C2 = C2_TX_COMPLETING;
	}
	// End Pedro R
}	}
}	}
#else	#else
if (UART1_S1 & UART_S1_RDRF) {	if (UART1_S1 & UART_S1_RDRF) {
Line 582	**Line 635**
}	}
}	}
c = UART1_C2;	c = UART1_C2;
if ((c & UART_C2_TIE) && (UART1_S1 & UART_S1_TDRE)) {	if ((c & UART_C2_TIE) && (UART1_S1 & UART_S1_TDRE)) { // data buffer empty
head = tx_buffer_head;	head = tx_buffer_head;
tail = tx_buffer_tail;	tail = tx_buffer_tail;
if (head == tail) {	if (head == tail) { // PedroR: head == tail is the same as my TX_BUFFER_EMPTY macro
UART1_C2 = C2_TX_COMPLETING;	// PedroR
	if (queueing_tx_data) {
	UART1_C2 = C2_TX_INACTIVE; // the code block protected by queueing_tx_data will address UART_C2
	} else if (!TX_BUFFER_EMPTY) {
	UART1_C2 = C2_TX_ACTIVE; // force interrupts to continue firing
	} else {
	UART1_C2 = C2_TX_COMPLETING;
	}
	// End Pedro R
} else {	} else {
if (++tail >= SERIAL2_TX_BUFFER_SIZE) tail = 0;	if (++tail >= SERIAL2_TX_BUFFER_SIZE) tail = 0;
n = tx_buffer[tail];	n = tx_buffer[tail];
Line 596	**Line 657**
}	}
}	}
#endif	#endif
if ((c & UART_C2_TCIE) && (UART1_S1 & UART_S1_TC)) {	if ((c & UART_C2_TCIE) && (UART1_S1 & UART_S1_TC)) { // TX complete
transmitting = 0;	
if (transmit_pin) transmit_deassert();	
UART1_C2 = C2_TX_INACTIVE;	
	// PedroR
	if (queueing_tx_data) {
	UART1_C2 = C2_TX_INACTIVE; // the code block protected by queueing_tx_data will address UART_C2
	} else if (!TX_BUFFER_EMPTY) {
	UART1_C2 = C2_TX_ACTIVE; // force interrupts to continue firing
	} else {
	transmitting = 0;
	if (transmit_pin) transmit_deassert();
	UART1_C2 = C2_TX_INACTIVE;
	}
	if (*debug_pin == 0) { *debug_pin = 1; } else { *debug_pin = 0; }
	// End PedroR
}	}
}	}
}	}