**i2c\_driver.c**

1. **i2c\_init()**

* This function initialises the i2c modules with the max. no. of i2c modules present.
* Currently we are using the “Vajra” platform which has two I2C instances having base addresses as “0x00040000” and “0x00041400”.

void i2c\_init()

{

for(int i=0; i< MAX\_I2C\_COUNT; i++)

{

i2c\_instance[i] = (i2c\_struct\*) (I2C0\_BASE + (i \* I2C\_OFFSET));

}

}

1. **config\_i2c()**

* This routine configures the serial clock frequency count and prescaler count.
* There are 4 registers which are configurable.
* This function writes into the register based on the passed address. The serial clock count and prescaler count decides the frequency (sck) that needs to be used for the I2C serial communication. Then it resets the status register.
* The function returns 0 if successful, else it returns -ENXIO.

int config\_i2c(i2c\_struct \* instance, unsigned char prescale\_div, unsigned char scl\_div)

{

unsigned char temp = 0;

log\_debug("\tI2C: Initializing the Controller\n");

if(prescale\_div != instance->prescale )

{

instance->prescale = prescale\_div; // Initializing Prescaler

}

if(scl\_div != instance->scl )

{

instance->scl = scl\_div;

// Setting the I2C clock value to be 1, which will set the clock for module and

// prescaler clock

}

instance->control = I2C\_PIN;

// Waiting for a specified time

waitfor(900);

// 1 Second software wait -- Should be 900000 but setting to 900 now since simulation

// is already slow

/\* Enable Serial Interface \*/

instance->control = I2C\_IDLE;

waitfor(900);

/\* 1 Second software wait -- Should be 900000 but setting to 900 now since simulation

is already slow \*/

temp = instance->status;

/\* Check to see if I2C is really in Idle and see if we can access the status register -- If

not something wrong in initialization. This also verifies if Control is properly written

since zero bit will be initialized to zero \*/

if(temp != (I2C\_PIN | I2C\_BB))

{

return -ENXIO;

}

return 0;

}

1. **wait\_till\_I2c\_bus\_free()**

* This function waits for the i2c bus to be free.
* Once an I2C Transaction is started, the bus needs to be freed for other devices to use the bus. This function checks the busy bit in the status register to become free for a particular time. If it becomes free, returns 0, else negative value.
* The function returns 0 if the bus becomes free, else returns ETIMEDOUT.

int wait\_till\_I2c\_bus\_free(i2c\_struct \* instance)

{

int timeout = DEF\_TIMEOUT; // macros defined as 60

int status;

status = instance->status;

while (!(status & I2C\_BB) && --timeout)

{

waitfor(20000); /\* wait for 100 us \*/

status = instance->status;

}

if (timeout == 0)

{

return ETIMEDOUT; // Bus busy wait - timed out.

}

return 0;

}

1. **wait\_till\_txrx\_operation\_Completes()**

* This function waits in the loop till the i2c tx/rx operation completes
* The PIN bit in the status register becomes high when the tx/rx operation starts and becomes low once done. This function checks whether the tx/rx operation is complete or not.
* The function returns zero on success, else returns -ETIMEOUT

int wait\_till\_txrx\_operation\_Completes(i2c\_struct \* instance, int \*status)

{

int timeout = DEF\_TIMEOUT; // macros defined as 60

\*status = instance->status;

while ((\*status & I2C\_PIN) && --timeout)

{

waitfor(10000); /\* wait for 100 us \*/

\*status = instance->status;

}

if (timeout == 0)

{

return ETIMEDOUT;

}

waitfor(10000); /\* wait for 100 us \*/

return 0;

}

1. **sendbytes()**

* This function is called when the user wants to write ‘n’ number of bits over an i2c bus.
* This function returns the number of bytes written, else EREMOTEIO.
* The parameters passed are-
  + i2c\_struct\* instance --- i2c structure pointer
  + const char \*buf --- Pointer to buffer which contains the data to be written.
  + int count --- No of bytes to be written.
  + int last --- If 1, I2C stop command can be sent
  + int eni --- External Interrupt Control

int sendbytes(i2c\_struct \* instance, const char \*buf, int count, int last, int eni)

{

int wrcount, status, timeout;

printf("\tStarting Write Transaction -- Did you create tri1 nets for SDA and SCL in

verilog?\n");

for (wrcount=0; wrcount<count; ++wrcount)

{

instance->data = buf[wrcount];

timeout = wait\_till\_txrx\_operation\_Completes(instance, &status);

if (timeout)

{

printf("\tTimeout happened - Write did not go through the BFM -- Diagnose\n");

instance->control = I2C\_STOP;

return EREMOTEIO;

}

if (status & I2C\_LRB)

{

instance->control = I2C\_STOP;

printf("\tSome status check failing\n");

return EREMOTEIO;

}

}

if (last)

{

printf("\tLast byte sent : Issue a stop\n");

instance->control = I2C\_STOP;

}

else

{

printf("\tSending Rep Start and doing some other R/W transaction\n");

if(!eni)

instance->control = I2C\_REPSTART;

else

instance->control = I2C\_REPSTART\_ENI;

}

return wrcount;

}

1. **readbytes()**

* This function reads the ‘n’-number of bytes over the I2C Bus and stores the same in the "buf" pointer.
* The parameters passed are-
  + i2c\_struct\* instance--- i2c structure pointer
  + const char \*buf --- Pointer to buffer which contains the read data.
  + int count --- No of bytes to be read.
  + int last --- If 1, I2C stop command can be sent
* The function returns the number of bytes read over the i2c bus, else -1.

int readbytes(i2c\_struct \* instance, char \*buf, int count, int last)

{

int i, status;

int wfp;

/\* increment number of bytes to read by one -- read dummy byte \*/

for (i = 0; i <= count; i++)

{

wfp = wait\_till\_txrx\_operation\_Completes(instance, &status);

if (wfp) {

instance->control = I2C\_STOP;

return -1;

}

if ((status & I2C\_LRB) && (i != count)) {

instance->control = I2C\_STOP;

printf("\tNo ack\n");

return -1;

}

if (i)

{

buf[i - 1] = instance->data;

printf("\n Read Value: %x", buf[i - 1]);

}

else

instance->data = !instance->data; /\* dummy read \*/

if (i == count - 1) {

instance->control = I2C\_ESO;

}

else if (i == count)

{

if (last)

instance->control = I2C\_STOP;

else

instance->control = I2C\_REPSTART\_ENI;

}

}

return i-1; //excluding the dummy read

}

1. **i2c\_send\_slave\_address()**

* This function writes a slave address into the i2b to start the write or read operation.
* The parameters passed are-
  + i2c\_struct\* instance--- i2c structure pointer
  + unsigned char slaveAddress
  + unsigned char rdWrCntrl
  + unsigned long delay
* This function returns Zero if success; else non zero

int i2c\_send\_slave\_address(i2c\_struct \* instance, unsigned char slaveAddress, unsigned char rdWrCntrl, unsigned long delay)

{

int timeout;

unsigned char temp = 0;

int status = 0;

delay = delay;

if(rdWrCntrl == 0)

slaveAddress |= I2C\_WRITE;

else

slaveAddress |= I2C\_READ;

//Writing the slave address that needs to be written into the data register.

instance->data = slaveAddress;

//Waits till the bus becomes free.

while(wait\_till\_I2c\_bus\_free(instance))

{

return EI2C\_BUS\_ERROR; // Error in Waiting for BB

}

//Send the start condition and slave address to slave

#ifndef USE\_SA\_WRITE\_I2C\_INTERRUPT

instance->control = I2C\_START; //Sending the slave address to the I2C slave

waitfor(900);

//Wait for PIN to become low.

timeout = wait\_till\_txrx\_operation\_Completes(instance, &status);

if (timeout)

{

//Asking the controller to send a start signal to initiate the transaction

printf("\tTimeout happened - Write did not go through the BFM -- Diagnose\n");

instance->control = I2C\_STOP;

return EI2C\_PIN\_ERROR;

}

if (status & I2C\_LRB)

{

instance->control = I2C\_STOP;

printf("\tSome status check failing\n");

return EI2C\_LRB\_ERROR;

}

#else

i2c\_complete\_flag = 0;

instance->control = I2C\_START\_ENI; //Sending the slave address to the I2C slave

while(!i2c\_complete\_flag);

log\_info("\n Slave Address Write Operation is complete.");

i2c\_complete\_flag = 0;

#endif

return I2C\_SUCCESS;

}

1. **i2c\_write\_data()**

* This function writes one byte of data to the slave I2C DEVICE.
* The parameters passed are-
  + i2c\_struct\* instance--- i2c structure pointer
  + unsigned char writedata
  + unsigned char delay
* The function returns Zero on success, else it returns -EREMOTEIO

int i2c\_write\_data(i2c\_struct \* instance, unsigned char writeData, unsigned char delay)

{

int timeout;

int status = 0;

delay = delay;

instance->data= writeData;

#ifndef USE\_WRITE\_I2C\_INTERRUPT

timeout = wait\_till\_txrx\_operation\_Completes(instance, &status);

if (timeout)

{

printf("\tTimeout happened - Write did not go through the BFM -- Diagnose\n");

instance->control = I2C\_STOP;

return EREMOTEIO;

}

if (status & I2C\_LRB)

{

instance->control = I2C\_STOP;

printf("\tSome status check failing\n");

return EI2C\_LRB\_ERROR;

}

#else

i2c\_complete\_flag = 0;

instance->control = I2C\_STOP\_ENI; //Sending the slave address to the I2C slave

while(!i2c\_complete\_flag);

log\_info("\n Write Operation is complete.");

i2c\_complete\_flag = 0;

#endif

return I2C\_SUCCESS;

}

1. **i2c\_read\_data()**

* This function reads a byte of data over the I2C bus from the passed I2C location.
* The parameters passed are-
  + i2c\_struct\* instance--- i2c structure pointer
  + unsigned char \*read\_data
  + unsigned char delay
* The function returns Zero on success, else it returns -ETIMEOUT

int i2c\_read\_data(i2c\_struct \* instance, unsigned char \*read\_data, unsigned char delay)

{

int status = 0;

/\* Make a dummy read as per spec of the I2C controller \*/

\*read\_data = instance->data;

#ifdef USE\_WRITE\_I2C\_INTERRUPT

i2c\_complete\_flag = 0;

instance->control = I2C\_REPSTART\_ENI; //~

while(!i2c\_complete\_flag);

\*read\_data = instance->data;

printf("\n I2C Read Data = %x", i2c\_read\_data);

#else

while(wait\_till\_txrx\_operation\_Completes(instance, &status))

{

printf("\twaiting for pin\n");

waitfor(delay);

}

#endif

return I2C\_SUCCESS;

}

1. **i2c\_read\_data\_nack()**

* This function reads a byte of data over the I2C bus from the passed I2C location. Then sends the request in the next byte.
* The parameters passed are-
  + i2c\_struct\* instance--- i2c structure pointer
  + unsigned char \*read\_data
  + unsigned char delay
* The function returns Zero on success else it will return -ETIMEOUT

int i2c\_read\_data\_nack(i2c\_struct \* instance, unsigned char \*read\_data, unsigned char delay)

{

/\* Make a dummy read as per spec of the I2C controller \*/

\*read\_data = instance->data; //~

delay=delay;

#ifdef USE\_WRITE\_I2C\_INTERRUPT

i2c\_complete\_flag = 0;

instance->control = I2C\_REPSTART\_ENI;

while(!i2c\_complete\_flag);

\*read\_data = instance->data;

printf("\n I2C Read Data = %x", i2c\_read\_data);

#endif

printf("\n I2C Read Data = %x", \*read\_data);

return I2C\_SUCCESS;

}

1. **i2c\_send\_interrupt\_slave\_address()**

* This is an Interrupt based routine to send slave address to the I2C slave device
* The parameters passed are-
  + i2c\_struct\* instance--- i2c structure pointer
  + unsigned char slaveAddress
  + unsigned char rdWrCntrl
  + unsigned long delay
* The function returns Zero on success, else corresponding error value.

int i2c\_send\_interrupt\_slave\_address(i2c\_struct \* instance, unsigned char slaveAddress, unsigned char rdWrCntrl, unsigned long delay)

{

int timeout;

unsigned char temp = 0;

int status = 0;

delay = delay;

if(rdWrCntrl == 0)

slaveAddress |= I2C\_WRITE;

else

slaveAddress |= I2C\_READ;

//Writing the slave address that needs to be written into data register.

instance->data = slaveAddress;

//Waits till the bus becomes free.

while(wait\_till\_I2c\_bus\_free(instance))

{

return EI2C\_BUS\_ERROR; //Error in Waiting for BB

}

//Send the start condition and slave address to slave

#ifndef USE\_SA\_WRITE\_I2C\_INTERRUPT

instance->control = I2C\_START;

//Sending the slave address to the I2C slave

//Wait for PIN to become low.

timeout = wait\_till\_txrx\_operation\_Completes(instance, &status);

if (timeout)

{

//Asking the controller to send a start signal to initiate the transaction

printf("\tTimeout happened - Write did not go through the BFM -- Diagnose\n");

instance->control = I2C\_STOP;

return EI2C\_PIN\_ERROR;

}

if (status & I2C\_LRB)

{

instance->control = I2C\_STOP;

printf("\tSome status check failing\n");

return EI2C\_LRB\_ERROR;

}

#else

i2c\_complete\_flag = 0;

instance->control = I2C\_REPSTART\_ENI;

//Sending the slave address to the I2C slave

while(!i2c\_complete\_flag);

i2c\_complete\_flag = 0;

#endif

return I2C\_SUCCESS;

}

1. **i2c\_read\_interrupt\_data()**

* It is an interrupt-based I2C read function to read from the I2C slave.
* The parameters passed are-
  + i2c\_struct\* instance--- i2c structure pointer
  + unsigned char \*read\_data
  + unsigned char delay
  + unsigned char last
* This function returns Zero on success.

int i2c\_read\_interrupt\_data(i2c\_struct \* instance, unsigned char \*read\_data, unsigned char delay, unsigned char last)

{

int status = 0;

/\* Make a dummy read as per spec of the I2C controller \*/

\*read\_data = instance->data;

#ifdef USE\_READ\_I2C\_INTERRUPT

i2c\_complete\_flag = 0;

if(last)

{

instance->control = I2C\_STOP\_ENI; //~

while(!i2c\_complete\_flag);

}

else

{

/\* Needs to be tested \*/

// instance->control = I2C\_REPSTART\_ENI;

// printf("\n Call I2C rep. start eni");

// while(!i2c\_complete\_flag);

}

printf("\n I2C Read Data = %x", \*read\_data);

#else

while(wait\_till\_txrx\_operation\_Completes(instance, &status))

{

printf("\twaiting for pin\n");

waitfor(delay);

}

if(!last)

{

printf("\n Rep Start");

// instance->control = I2C\_REPSTART;

}

else

{

printf("\nCall I2C Stop");

instance->control = I2C\_STOP;

}

#endif

return I2C\_SUCCESS;

}

1. **i2c\_write\_interrupt\_data()**

* This function writes a byte of data into a slave I2C bus using interrupt.
* The parameters passed are-
  + i2c\_struct\* instance--- i2c structure pointer
  + unsigned char writeData
  + unsigned char delay
  + unsigned char last
* This function returns Zero on success. Else based on the error.

int i2c\_write\_interrupt\_data(i2c\_struct \* instance, unsigned char writeData, unsigned char delay, unsigned char last)

{

int timeout;

int status = 0;

delay = delay;

instance->data = writeData;

#ifndef USE\_WRITE\_I2C\_INTERRUPT

timeout = wait\_till\_txrx\_operation\_Completes(instance, &status);

if (timeout) {

printf("\tTimeout happened - Write did not go through the BFM -- Diagnose\n");

instance->control = I2C\_STOP; //~

return EREMOTEIO;

}

if (status & I2C\_LRB)

{

instance->control = I2C\_STOP;

printf("\tSome status check failing\n");

return EI2C\_LRB\_ERROR;

}

if(1 == last)

{

instance->control = I2C\_STOP;;

printf("\tI2C Write Success and completes\n");

}

#else

i2c\_complete\_flag = 0;

if(last)

{

instance->control = I2C\_STOP\_ENI; //Sending the sslave address to the I2C slave

printf("\n Calling stop eni write");

while(!i2c\_complete\_flag);

}

else

{

// instance->control = I2C\_REPSTART\_ENI;

// printf("\n Calling repstart eni write");

// while(!i2c\_complete\_flag);

}

log\_info("\n Write Operation is complete.");

i2c\_complete\_flag = 0;

#endif

return I2C\_SUCCESS;

}