Konfigurierbare eingebettete Systeme

Beuth Hochschule - Wintersemester 2018/19

Laborübung 4

Gruppe *Index*
Omid Rahimian Mashhadi Mat.Nr.: 872958
Torsten Michael Schenk Mat.Nr.: 838995

Abgabedatum: 07.01.2019

Seitenanzahl: 9

Inhaltsverzeichnis

1 Vorwort

Bei der Recherche zur Bearbeitung der Übungen wurden viele englischsprachige Webseiten zu rate gezogen. Generell kann man sagen, dass englische Fachbegriffe sich im Bereich FPGA und embedded Design etabliert haben, so dass eine Übersetzung eher verwirren als helfen würde. Daher haben wir uns entschieden, die **englischen** Bezeichner und Beschreibungen beizubehalten.
Um Codeabschnitte besser von Beschreibungen besser unterscheiden zu können, wurde eine eigene Schriftart verwendet:

```
Kommandozeilen Eingaben und Codesnippets werden wie HIER dargestellt.
```

2 Aufgabe 1

In der Laborübung wurde ein IP Core, welcher in C Code beschrieben wurde, durch HLS eingesetzt.

2.1 HLS Theorie

High-Level-Synthese (HLS) verbindet Hardware- und Software-Domänen zusammen. Entwickler können so die Vorteile der Hardwareimplementierung direkt aus den Verhaltensweisen des Algorithmus ziehen, die in C-ähnlichen Sprachen mit hohem Abstraktionsgrad angegeben werden. Um die Leistungslücke zwischen den manuellen und HLS-basierten FPGA-Designs zu verringen, werden in den heutigen HLS-Werkzeugen verschiedene Code-Optimierungsformen bereitgestellt.

2.2 Beschreibung

Das Musterdesign ist ein FIR-Filter. Das muss in HLS-Software synthetisiert und als IP gespeichert werden.

2.3 Die Zahl der Zyklen von dem Code

Die Latenzverzögerung ist auf die RTL Logik zurückzuführen, die aus der mit Shift_Accum_Loop bezeichneten Schleife synthetisiert wird. Die Schleife wird 11 mal ausgeführt (Trip Count). Jede Ausführung erfordert 7 Taktzyklen (Iteration Latency) für insgesamt 77 Taktzyklen, um alle Iterationen dieser Schleife synthetisierten Logik (Latency) auszuführen.
7 * 11 = 77

⊟ **Detail**

⊞ **Instance**

⊟ **Loop**

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Shift_Accum_Loop	77	77	7	-	-	11	no

Leistungsschätzung

2.4 Die benötigten Ressourcen

Der Entwurf verwendet einen einzelnen Speicher, der als LUTRAM implementiert ist, 4 DSP48s und 243 Flip-Flops und LUTs.

Synthesis(solution1)

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	39
FIFO	-	-	-	-
Instance	-	4	0	0
Memory	0	-	64	6
Multiplexer	-	-	-	114
Register	-	-	179	-
Total	0	4	243	159
Available	650	600	202800	101400
Utilization (%)	0	~0	~0	~0

Detail

Instance

Instance	Module	BRAM_18K	DSP48E	FF	LUT
fir_mul_32s_32s_32_3_U0	fir_mul_32s_32s_32_3	0	4	0	0
Total	1	0	4	0	0

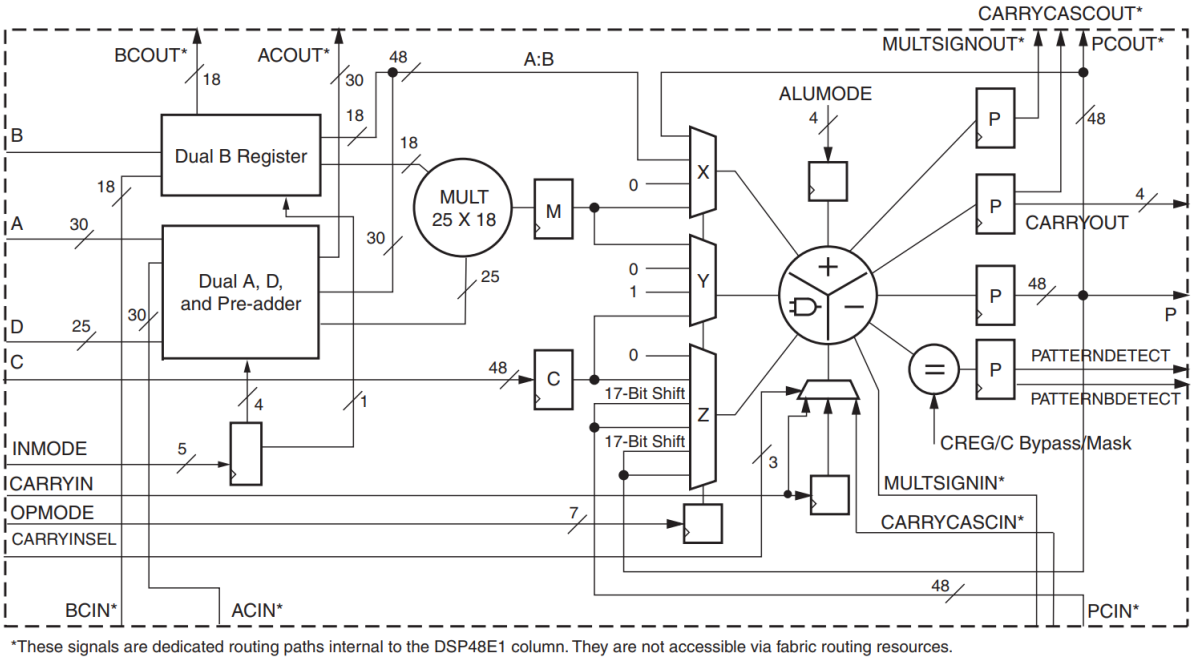
Nutzungsschätzungen

2.5 Eine Multiplikation mit vier DSP48E

Da es sich bei den Daten um einen C-Integer-Typ handelt, der 32-Bit ist. Dies führt zu großen Multiplizierern.Ein DSP48-Multiplizierer ist 18 Bit und erfordert mehrere DSP48, um eine Multiplikation für Datenbreiten von mehr als 18 Bit zu implementieren.

```
                                    [mHigh mLow]    X
                                    [nHigh nLow]
                                    ------------
                          [mHigh * nLow] [mLow * nLow]
    [mHigh * nHigh] [mLow * nHigh]
-------------------------------------------------------------------
[mHigh * nHigh] + [mLow * nHigh + mHigh * nLow] + [mLow * nLow]
-------------------------------------------------------------------
```

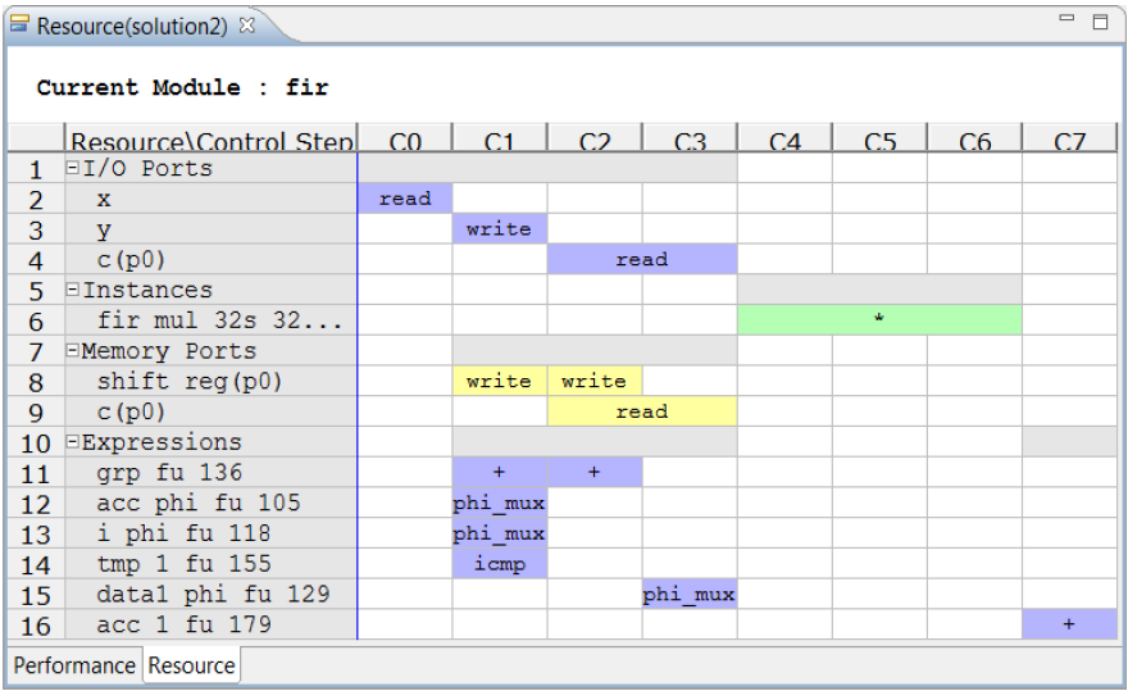
Multiplikationsformel



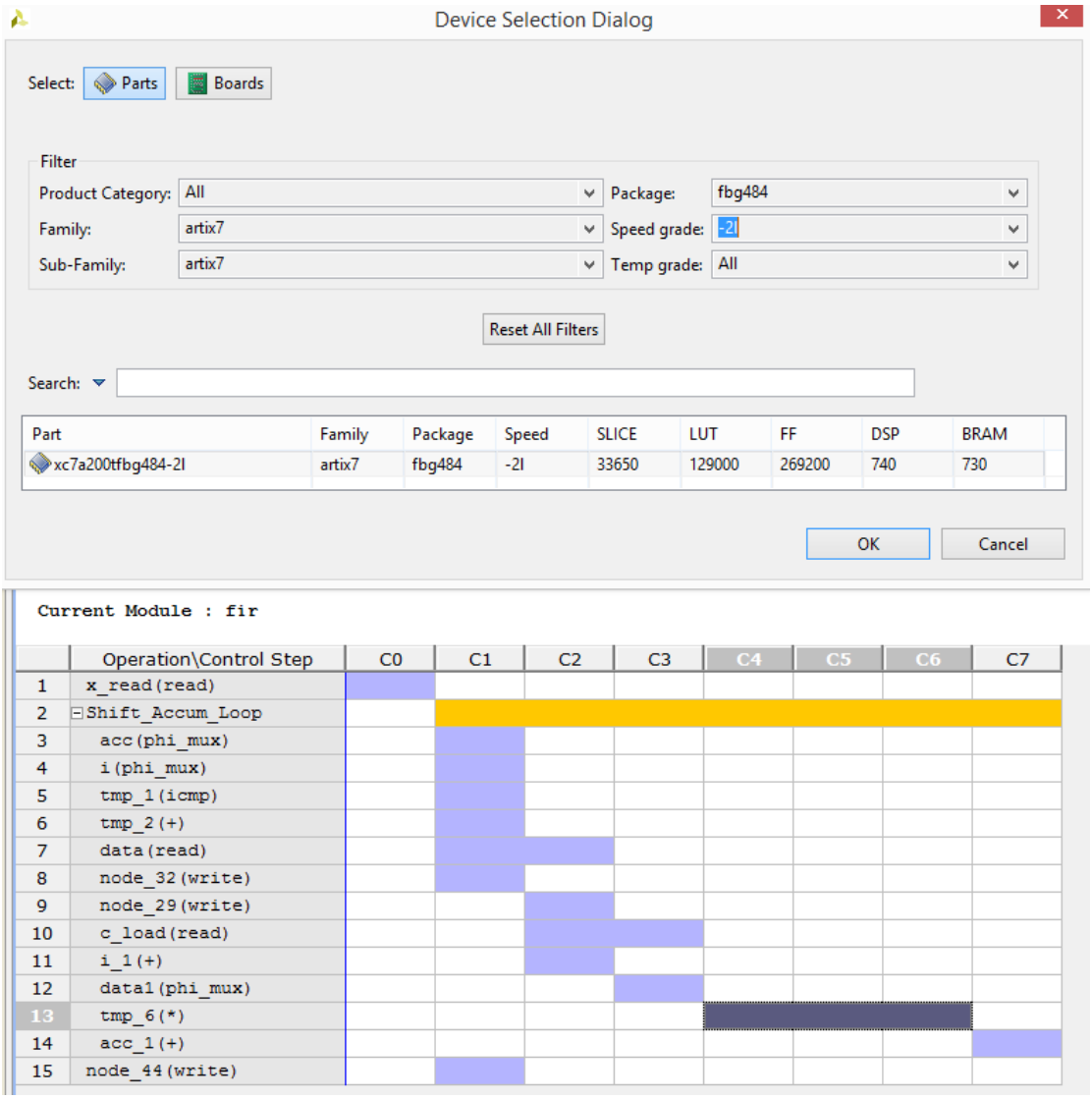
DSP48e

2.6 die benötigten Operationen des Design und deren Laufzeit

In der linken Feld der PerformanceAnsicht werden die Operationen im Modul der RTLHierarchie angezeigt.



Quellen



Performanz

3 Aufgabe 2

In der Aufgabe 2 wurde eine Fibonacci-Algorithm in c mitgebracht. Der Code wurde mit HLS synthetisiert und sowohl den Ressourcenverbrauch als auch das Timing unseres Codes analysiert.

3.1 C-Code

C-Code zur Berechnung des n-ten Wertes F(n) der Fibonacci-Folge aus den Startwerten F0 = 0 und F1 = 1

Fibonacci für Hardware Implementation

```
#include "fir.h"

void fir (
  data_t result[20],
  data_t n
```

```
  ) {

  int i;
  *result = 0;
  *(result+1) = 1;
  Shift_Accum_Loop: for (i=2;i<20;i++) {
  *(result+i) =  *(result+i-1) +  *(result+i-2);
  }
}
```

Testbench

```
#include <stdio.h>
#include <math.h>
#include "fir.h"

int main () {
   FILE         *fp;

  data_t signal, output[N];
  //coef_t taps[N] = {2,100,9};

  int i;

  fp=fopen("out.dat","w");
  fir(output,N);

  for (i=0;i<N;i++) {
// Execute the function with latest input
// Save the results.
    fprintf(fp," the %d-Fibonatchi is =  %d\n",i+1,output[i]);
  }
  fclose(fp);

  printf ("Comparing against output data \n");
  if (system("diff -w out.dat out.gold.dat")) {

fprintf(stdout, "*******************************************\n");
fprintf(stdout, "FAIL: The result is not correct\n");
fprintf(stdout, "*******************************************\n");
     return 1;
  } else {
fprintf(stdout, "*******************************************\n");
fprintf(stdout, "PASS: The result is correct!\n");
fprintf(stdout, "*******************************************\n");
```

```
    return 0;
  }
}
```

3.2 C-Code Analyse

Das Bild zeigt den Ressourcenverbrauch und das Timing

Synthesis(solution1) Performance(solution1)

Current Module : fir

	Operation\Control Step	C0	C1	C2
1	n_read(read)			
2	node_8(write)			
3	node_10(write)			
4	Shift_Accum_Loop			
5	i(phi_mux)			
6	tmp(icmp)			
7	sum(+)			
8	result_load(read)			
9	sum3(+)			
10	result_load_1(read)			
11	tmp_2(+)			
12	node_32(write)			
13	i_1(+)			

Perfomance in Zyklen

	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type
fir	0	0	8	97	37	38	none

Performance Profile Resource Profile

	BRAM	DSP	FF	LUT	Bits P0	Bits P1	Bits P2	Banks/Depth	Words	W*Bits*Banks
fir	0	0	8	97						
I/O Ports(2)					64					
Instances(0)	0	0	0	0						
Memories(0)	0		0	0	0			0	0	0
Expressions(5)	0	0	0	49	52	43	0			
+	0	0	0	47	47	38	0			
sum3_fu	0	0	0	5	5	3	0			
sum_fu	0	0	0	5	5	2	0			
i_1_fu_1	0	0	0	5	5	1	0			
tmp_1_f	0	0	0	32	32	32	0			
icmp	0	0	0	2	5	5	0			
exitcon	0	0	0	2	5	5	0			
Registers(2)			8		8					
ap_CS_fsm			3		3					
i_reg_85			5		5					
Channels(0)	0		0	0	0			0	0	0
Multiplexers(5)	0		0	48	48			0		
result_d0	0		0	32	32			0		
result_addr	0		0	5	5			0		
result_addr	0		0	5	5			0		
ap_NS_fsm	0		0	1	1			0		
i_reg_85	0		0	5	5			0		

Date: Mon Jan 07 17:39:06 2019
Version: 2016.2 (Build 1577090 on Thu Jun 02 16:59:10 MDT
Project: Selbst_fir_prj
Solution: solution1
Product family: zynq
Target device: xc7z020clg484-1

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	7.22	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
37	37	38	38	none

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	49
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	-	-	-	-
Multiplexer	-	-	-	48
Register	-	-	8	-
Total	0	0	8	97
Available	280	220	106400	53200
Utilization (%)	0	0	~0	~0

Detail

Ressourcenverbrauch 1

Expression

Variable Name	Operation	DSP48E	FF	LUT	Bitwidth P0	Bitwidth P1
i_1_fu_137_p2	+	0	0	5	5	1
sum3_fu_114_p2	+	0	0	5	5	3
sum_fu_103_p2	+	0	0	5	5	2
tmp_1_fu_130_p2	+	0	0	32	32	32
exitcond_fu_97_p2	icmp	0	0	2	5	5
Total	5	0	0	49	52	43

Multiplexer

Name	LUT	Input Size	Bits	Total Bits
ap_NS_fsm	1	4	1	4
i_reg_85	5	2	5	10
result_address0	5	4	5	20
result_address1	5	3	5	15
result_d0	32	3	32	96
Total	48	16	48	145

Register

Name	FF	LUT	Bits	Const Bits
ap_CS_fsm	3	0	3	0
i_reg_85	5	0	5	0
Total	8	0	8	0

Ressourcenverbrauch 2